1. Create a nd array of size 3 x 3 x 3 and perform subtraction operation as per x axis, y axis and z axis.

```
import numpy as np
y=np.array([[[1,2,3],
             [4,5,6],
             [7,8,9]],
            [[10, 11, 12],
             [13,14,15],
             [16,17,18]],
            [[19,20,21],
             [22,23,24],
             [25,26,27]]])
y
```

```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[19, 20, 21],
        [22, 23, 24],
        [25, 26, 27]]])
```

NumPy arange() is one of the array creation routines based on numerical ranges. It creates an instance of ndarray with evenly spaced values and returns the reference to it.

```
d=np.arange(16)
d
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

An ndarray object has many methods which operate on or with the array in some fashion, typically returning an array result.

```
arr = np.array([[0, 1], [1, 1], [2, 2]])
rowsum = arr.sum(-1)
arr[rowsum <= 2, :]
```

```
array([[0, 1],
       [1, 1]])
```

```
y.shape
```

```
(3, 3, 3)
```

```
### 1. Create a nd array of size 3 x 3 x 3 and perform subtraction operation as per x axis, y axis and z axis.
# x axis
x_sub = np.diff(y,axis=0)
print(x_sub)
```

```
[[[9 9 9]
  [9 9 9]
  [9 9 9]]

 [[9 9 9]
  [9 9 9]
  [9 9 9]]]
```

```
# y axis
y_sub = np.diff(y, axis=1)
print(y_sub)
```

```
[[[3 3 3]
  [3 3 3]]

 [[3 3 3]
  [3 3 3]]

 [[3 3 3]
  [3 3 3]]]
```

```
# z axis
z_sub =np.diff(y, axis=2)
print(z_sub)
```

```
[[[1 1]
  [1 1]
  [1 1]]

 [[1 1]
  [1 1]
  [1 1]]

 [[1 1]
  [1 1]
  [1 1]]]
```

2. Demonstrate the following on the above created nd array:

1. basic indexing
2. slicing
3. boolean indexing
4. fancy indexing

1. basic indexing

```
x=np.arange(15)
x.shape=(3,5)
x
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
```

```
[10, 11, 12, 13, 14]])
```

```
x[2,-4]
```

```
11
```

```
x[0]
```

```
array([0, 1, 2, 3, 4])
```

```
x[1][2]
```

```
7
```

2. Slicing

```
x=np.array([1,35,67,83,92,73,63,99,100])
y=x[1:7:2]
y
```

```
array([35, 83, 73])
```

```
x[2:6:2]
```

```
array([67, 92])
```

```
x[4:7]
```

```
array([92, 73, 63])
```

3. Boolean indexing this is advanced indexing is used array as object of boolean type, such may use a comparitional operators. A common use case for this is filtering for desired value elements.

```
#For example, one may wish to select all entries from an array which are not NaN:
x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])
x[~np.isnan(x)]
```

```
array([1., 2., 3.])
```

numpy. isnan() returns a Boolean array. It returns True if an element is NaN . It returns False otherwise.

```
import numpy as np
t = np.array([[[1],[3],[5]], [[7],[9],[11]]])
t.shape
```

```
(2, 3, 1)
```

```
y=t[1:2]
y
```

```
array([[[ 7],
        [ 9],
        [11]]])
```

```
x = np.arange(25).reshape(5, 5)
b = x > 20
b[:, 3]
```

```
array([False, False, False, False,  True])
```

```
#From an array, select all rows which sum up to less or equal two:
x = np.array([[0, 1], [1, 1], [2, 2]])
rowsum = x.sum(-1)
x[rowsum <= 2, :]
```

```
array([[0, 1],
       [1, 1]])
```

Fancy indexing is conceptually simple, it means a passing an array of indices together to access multiple array at once.

```
rand = np.random.RandomState(42)
#eturns a random integer value between the two lower and higher limits (including both limits) provided as two parameters
x = rand.randint(300, size=10)
print(x)
```

```
[102 270 106  71 188  20 102 121 214  87]
```

```
[x[5],x[2],x[4],x[3]]
```

```
[20, 106, 188, 71]
```

3. Demonstrate the use of the following universal functions:

1. Trigonometric functions (atleast 2)
2. Statistical Functions (atleast 2)
3. Bit-twiddling Functions (atleast 2)

Trigonometric functions functions work on radians, so angles need to be converted to radians by multiplying by pi/180. Only then we can call trigonometric functions. They take an array as input arguments.

```
# Python code to demonstrate trigonometric function
import numpy as np

# create an array of angles
angles = np.array([0, 30, 45, 60, 90, 180])

# conversion of degree into radians
```

```
# using deg2rad function
radians = np.deg2rad(angles)
print(radians)
```

```
[0.         0.52359878 0.78539816 1.04719755 1.57079633 3.14159265]
```

```
# cosine of angles
print('cosine of angles in the array:')
cosine_value = np.cos(radians)
print(np.cos(radians))
```

```
cosine of angles in the array:
[ 1.00000000e+00  8.66025404e-01  7.07106781e-01  5.00000000e-01
  6.12323400e-17 -1.00000000e+00]
```

```
# inverse cosine of cosine values
print('Inverse cosine of cosine values:')
print(np.arccos(cosine_value))
```

```
Inverse cosine of cosine values:
[0.         0.52359878 0.78539816 1.04719755 1.57079633 3.14159265]
```

```
# hypot function demonstration
base = 67
height = 56
print('hypotenuse of right triangle is:')
print(np.hypot(base, height))
```

```
hypotenuse of right triangle is:
87.32124598286491
```

Statistical functions are used to calculate mean, median, variance, minimum of array elements.

It includes functions like-

amin, amax - returns minimum or maximum of an array or along an axis

ptp - returns range of values (maximum-minimum) of an array or along an axis

percentile(a, p, axis) - calculate pth percentile of array or along specified axis

```
# Python code demonstrate statistical function
import numpy as np

# construct a weight array
weight = np.arange(25)

# minimum and maximum
print('Minimum and maximum weight of the students: ')
print(np.amin(weight), np.amax(weight))

# range of weight i.e. max weight-min weight
print('Range of the weight of the students: ')
print(np.ptp(weight))

# percentile
```

```
print('Weight below which 70 % student fall: ')
print(np.percentile(weight, 70))

# variance
print('variance weight of the students : ')
print(np.var(weight))
```

```
Minimum and maximum weight of the students:
0 24
Range of the weight of the students:
24
Weight below which 70 % student fall:
16.799999999999997
variance weight of the students :
52.0
```

Bit-twiddling functions accept integer values as input arguments and perform bitwise operations on binary representations of those integers.

It include functions like-

bitwise_and - performs bitwise and operation on two array elements

bitwies_or - performs bitwise or operation on two array elements

bitwise_xor - performs bitwise xor operation on two array elements

```
even = np.array([0, 2, 4, 6, 8, 16, 32])
odd = np.array([1, 3, 5, 7, 9, 17, 33])

# bitwise_and
print('bitwise_and of two arrays: ')
print(np.bitwise_and(even, odd))

# bitwise_or
print('bitwise_or of two arrays: ')
print(np.bitwise_or(even, odd))

# bitwise_xor
print('bitwise_xor of two arrays: ')
print(np.bitwise_xor(even, odd))
```

```
bitwise_and of two arrays:
[ 0  2  4  6  8 16 32]
bitwise_or of two arrays:
[ 1  3  5  7  9 17 33]
bitwise_xor of two arrays:
[1 1 1 1 1 1 1]
```

4. Download the dataset of your choice and demonstrate the following operations on it:

1. Read the dataset in the form of array

2. Perform any 5 nd array operations on the above dataset (for ex. reshaping, ravel, transpose, vstack, concatenate etc.)

```
import pandas as pd
df=pd.read_csv("/content/homelessness (1).csv")
```

```
arr=df.to_numpy()
print(arr)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'East North Central' 'Indiana' 3776.0 1482.0 6695497]
 [15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
 [16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
 [17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
 [18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
 [19 'New England' 'Maine' 1450.0 1066.0 1339057]
 [20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
 [21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
 [22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
 [23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
 [24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
 [25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
 [26 'Mountain' 'Montana' 983.0 422.0 1060665]
 [27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
 [28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
 [29 'New England' 'New Hampshire' 835.0 615.0 1353465]
 [30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
 [31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
 [32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
 [33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
 [34 'West North Central' 'North Dakota' 467.0 75.0 758080]
 [35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
 [36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
 [37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
 [38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
 [39 'New England' 'Rhode Island' 747.0 354.0 1058287]
 [40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
 [41 'West North Central' 'South Dakota' 836.0 323.0 878698]
 [42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
 [43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
 [44 'Mountain' 'Utah' 1904.0 972.0 3153550]
 [45 'New England' 'Vermont' 780.0 511.0 624358]
 [46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
 [47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
 [48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
 [49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
 [50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
```

```
df.shape
```

```
(51, 6)
```

```
# .reshape() to make a copy with the desired shape.
arr.reshape(1,306)
```

```
print(arr)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'East North Central' 'Indiana' 3776.0 1482.0 6695497]
 [15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
 [16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
 [17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
 [18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
 [19 'New England' 'Maine' 1450.0 1066.0 1339057]
 [20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
 [21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
 [22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
 [23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
 [24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
 [25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
 [26 'Mountain' 'Montana' 983.0 422.0 1060665]
 [27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
 [28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
 [29 'New England' 'New Hampshire' 835.0 615.0 1353465]
 [30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
 [31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
 [32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
 [33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
 [34 'West North Central' 'North Dakota' 467.0 75.0 758080]
 [35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
 [36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
 [37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
 [38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
 [39 'New England' 'Rhode Island' 747.0 354.0 1058287]
 [40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
 [41 'West North Central' 'South Dakota' 836.0 323.0 878698]
 [42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
 [43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
 [44 'Mountain' 'Utah' 1904.0 972.0 3153550]
 [45 'New England' 'Vermont' 780.0 511.0 624358]
 [46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
 [47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
 [48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
 [49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
 [50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
```

```
#ravel
arr.ravel
print(arr)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
```

```
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'East North Central' 'Indiana' 3776.0 1482.0 6695497]
 [15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
 [16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
 [17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
 [18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
 [19 'New England' 'Maine' 1450.0 1066.0 1339057]
 [20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
 [21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
 [22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
 [23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
 [24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
 [25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
 [26 'Mountain' 'Montana' 983.0 422.0 1060665]
 [27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
 [28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
 [29 'New England' 'New Hampshire' 835.0 615.0 1353465]
 [30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
 [31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
 [32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
 [33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
 [34 'West North Central' 'North Dakota' 467.0 75.0 758080]
 [35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
 [36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
 [37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
 [38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
 [39 'New England' 'Rhode Island' 747.0 354.0 1058287]
 [40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
 [41 'West North Central' 'South Dakota' 836.0 323.0 878698]
 [42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
 [43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
 [44 'Mountain' 'Utah' 1904.0 972.0 3153550]
 [45 'New England' 'Vermont' 780.0 511.0 624358]
 [46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
 [47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
 [48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
 [49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
 [50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
```

```
#ravel function is used to return a contiguous array. This function returns a 1D array that contains the input elements.
arr.ravel
print(arr)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
```

```
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'East North Central' 'Indiana' 3776.0 1482.0 6695497]
 [15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
 [16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
 [17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
 [18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
 [19 'New England' 'Maine' 1450.0 1066.0 1339057]
 [20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
 [21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
 [22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
 [23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
 [24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
 [25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
 [26 'Mountain' 'Montana' 983.0 422.0 1060665]
 [27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
 [28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
 [29 'New England' 'New Hampshire' 835.0 615.0 1353465]
 [30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
 [31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
 [32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
 [33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
 [34 'West North Central' 'North Dakota' 467.0 75.0 758080]
 [35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
 [36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
 [37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
 [38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
 [39 'New England' 'Rhode Island' 747.0 354.0 1058287]
 [40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
 [41 'West North Central' 'South Dakota' 836.0 323.0 878698]
 [42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
 [43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
 [44 'Mountain' 'Utah' 1904.0 972.0 3153550]
 [45 'New England' 'Vermont' 780.0 511.0 624358]
 [46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
 [47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
 [48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
 [49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
 [50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
```

```
#transpose
np.transpose(arr).shape
```

```
(6, 51)
```

```
#vstack
stacked_data = np.vstack((df, df))
stacked_data
```

```
array([[0, 'East South Central', 'Alabama', 2570.0, 864.0, 4887681],
       [1, 'Pacific', 'Alaska', 1434.0, 582.0, 735139],
       [2, 'Mountain', 'Arizona', 7259.0, 2606.0, 7158024],
       [3, 'West South Central', 'Arkansas', 2280.0, 432.0, 3009733],
       [4, 'Pacific', 'California', 109008.0, 20964.0, 39461588],
       [5, 'Mountain', 'Colorado', 7607.0, 3250.0, 5691287],
       [6, 'New England', 'Connecticut', 2280.0, 1696.0, 3571520],
       [7, 'South Atlantic', 'Delaware', 708.0, 374.0, 965479],
       [8, 'South Atlantic', 'District of Columbia', 3770.0, 3134.0,
        701547],
```

```
[9, 'South Atlantic', 'Florida', 21443.0, 9587.0, 21244317],
[10, 'South Atlantic', 'Georgia', 6943.0, 2556.0, 10511131],
[11, 'Pacific', 'Hawaii', 4131.0, 2399.0, 1420593],
[12, 'Mountain', 'Idaho', 1297.0, 715.0, 1750536],
[13, 'East North Central', 'Illinois', 6752.0, 3891.0, 12723071],
[14, 'East North Central', 'Indiana', 3776.0, 1482.0, 6695497],
[15, 'West North Central', 'Iowa', 1711.0, 1038.0, 3148618],
[16, 'West North Central', 'Kansas', 1443.0, 773.0, 2911359],
[17, 'East South Central', 'Kentucky', 2735.0, 953.0, 4461153],
[18, 'West South Central', 'Louisiana', 2540.0, 519.0, 4659690],
[19, 'New England', 'Maine', 1450.0, 1066.0, 1339057],
[20, 'South Atlantic', 'Maryland', 4914.0, 2230.0, 6035802],
[21, 'New England', 'Massachusetts', 6811.0, 13257.0, 6882635],
[22, 'East North Central', 'Michigan', 5209.0, 3142.0, 9984072],
[23, 'West North Central', 'Minnesota', 3993.0, 3250.0, 5606249],
[24, 'East South Central', 'Mississippi', 1024.0, 328.0, 2981020],
[25, 'West North Central', 'Missouri', 3776.0, 2107.0, 6121623],
[26, 'Mountain', 'Montana', 983.0, 422.0, 1060665],
[27, 'West North Central', 'Nebraska', 1745.0, 676.0, 1925614],
[28, 'Mountain', 'Nevada', 7058.0, 486.0, 3027341],
[29, 'New England', 'New Hampshire', 835.0, 615.0, 1353465],
[30, 'Mid-Atlantic', 'New Jersey', 6048.0, 3350.0, 8886025],
[31, 'Mountain', 'New Mexico', 1949.0, 602.0, 2092741],
[32, 'Mid-Atlantic', 'New York', 39827.0, 52070.0, 19530351],
[33, 'South Atlantic', 'North Carolina', 6451.0, 2817.0, 10381615],
[34, 'West North Central', 'North Dakota', 467.0, 75.0, 758080],
[35, 'East North Central', 'Ohio', 6929.0, 3320.0, 11676341],
[36, 'West South Central', 'Oklahoma', 2823.0, 1048.0, 3940235],
[37, 'Pacific', 'Oregon', 11139.0, 3337.0, 4181886],
[38, 'Mid-Atlantic', 'Pennsylvania', 8163.0, 5349.0, 12800922],
[39, 'New England', 'Rhode Island', 747.0, 354.0, 1058287],
[40, 'South Atlantic', 'South Carolina', 3082.0, 851.0, 5084156],
[41, 'West North Central', 'South Dakota', 836.0, 323.0, 878698],
[42, 'East South Central', 'Tennessee', 6139.0, 1744.0, 6771631],
[43, 'West South Central', 'Texas', 19199.0, 6111.0, 28628666],
[44, 'Mountain', 'Utah', 1904.0, 972.0, 3153550],
[45, 'New England', 'Vermont', 780.0, 511.0, 624358],
[46, 'South Atlantic', 'Virginia', 3928.0, 2047.0, 8501286],
[47, 'Pacific', 'Washington', 16424.0, 5880.0, 7523869],
[48, 'South Atlantic', 'West Virginia', 1021.0, 222.0, 1804291],
[49, 'East North Central', 'Wisconsin', 2740.0, 2167.0, 5807406],
[50, 'Mountain', 'Wyoming', 434.0, 205.0, 577601],
[0, 'East South Central', 'Alabama', 2570.0, 864.0, 4887681],
[1, 'Pacific', 'Alaska', 1434.0, 582.0, 735139],
[2, 'Mountain', 'Arizona', 7259.0, 2606.0, 7158024],
[3, 'West South Central', 'Arkansas', 2280.0, 432.0, 3009733],
[4, 'Pacific', 'California', 109008.0, 20964.0, 39461588],
[5, 'Mountain', 'Colorado', 7607.0, 3250.0, 5691287]
```

```
#concatenate
concatenated_data = np.concatenate((df, df), axis=1)
concatenated_data
```

```
array([[0, 'East South Central', 'Alabama', 2570.0, 864.0, 4887681, 0,
        'East South Central', 'Alabama', 2570.0, 864.0, 4887681],
       [1, 'Pacific', 'Alaska', 1434.0, 582.0, 735139, 1, 'Pacific',
        'Alaska', 1434.0, 582.0, 735139],
       [2, 'Mountain', 'Arizona', 7259.0, 2606.0, 7158024, 2, 'Mountain',
        'Arizona', 7259.0, 2606.0, 7158024],
       [3, 'West South Central', 'Arkansas', 2280.0, 432.0, 3009733, 3,
        'West South Central', 'Arkansas', 2280.0, 432.0, 3009733],
       [4, 'Pacific', 'California', 109008.0, 20964.0, 39461588, 4,
        'Pacific', 'California', 109008.0, 20964.0, 39461588],
```

```
[5, 'Mountain', 'Colorado', 7607.0, 3250.0, 5691287, 5,
 'Mountain', 'Colorado', 7607.0, 3250.0, 5691287],
[6, 'New England', 'Connecticut', 2280.0, 1696.0, 3571520, 6,
 'New England', 'Connecticut', 2280.0, 1696.0, 3571520],
[7, 'South Atlantic', 'Delaware', 708.0, 374.0, 965479, 7,
 'South Atlantic', 'Delaware', 708.0, 374.0, 965479],
[8, 'South Atlantic', 'District of Columbia', 3770.0, 3134.0,
 701547, 8, 'South Atlantic', 'District of Columbia', 3770.0,
 3134.0, 701547],
[9, 'South Atlantic', 'Florida', 21443.0, 9587.0, 21244317, 9,
 'South Atlantic', 'Florida', 21443.0, 9587.0, 21244317],
[10, 'South Atlantic', 'Georgia', 6943.0, 2556.0, 10511131, 10,
 'South Atlantic', 'Georgia', 6943.0, 2556.0, 10511131],
[11, 'Pacific', 'Hawaii', 4131.0, 2399.0, 1420593, 11, 'Pacific',
 'Hawaii', 4131.0, 2399.0, 1420593],
[12, 'Mountain', 'Idaho', 1297.0, 715.0, 1750536, 12, 'Mountain',
 'Idaho', 1297.0, 715.0, 1750536],
[13, 'East North Central', 'Illinois', 6752.0, 3891.0, 12723071,
 13, 'East North Central', 'Illinois', 6752.0, 3891.0, 12723071],
[14, 'East North Central', 'Indiana', 3776.0, 1482.0, 6695497, 14,
 'East North Central', 'Indiana', 3776.0, 1482.0, 6695497],
[15, 'West North Central', 'Iowa', 1711.0, 1038.0, 3148618, 15,
 'West North Central', 'Iowa', 1711.0, 1038.0, 3148618],
[16, 'West North Central', 'Kansas', 1443.0, 773.0, 2911359, 16,
 'West North Central', 'Kansas', 1443.0, 773.0, 2911359],
[17, 'East South Central', 'Kentucky', 2735.0, 953.0, 4461153, 17,
 'East South Central', 'Kentucky', 2735.0, 953.0, 4461153],
[18, 'West South Central', 'Louisiana', 2540.0, 519.0, 4659690,
 18, 'West South Central', 'Louisiana', 2540.0, 519.0, 4659690],
[19, 'New England', 'Maine', 1450.0, 1066.0, 1339057, 19,
 'New England', 'Maine', 1450.0, 1066.0, 1339057],
[20, 'South Atlantic', 'Maryland', 4914.0, 2230.0, 6035802, 20,
 'South Atlantic', 'Maryland', 4914.0, 2230.0, 6035802],
[21, 'New England', 'Massachusetts', 6811.0, 13257.0, 6882635, 21,
 'New England', 'Massachusetts', 6811.0, 13257.0, 6882635],
[22, 'East North Central', 'Michigan', 5209.0, 3142.0, 9984072,
 22, 'East North Central', 'Michigan', 5209.0, 3142.0, 9984072],
[23, 'West North Central', 'Minnesota', 3993.0, 3250.0, 5606249,
 23, 'West North Central', 'Minnesota', 3993.0, 3250.0, 5606249],
[24, 'East South Central', 'Mississippi', 1024.0, 328.0, 2981020,
 24, 'East South Central', 'Mississippi', 1024.0, 328.0, 2981020],
[25, 'West North Central', 'Missouri', 3776.0, 2107.0, 6121623,
 25, 'West North Central', 'Missouri', 3776.0, 2107.0, 6121623],
[26, 'Mountain', 'Montana', 983.0, 422.0, 1060665, 26, 'Mountain',
 'Montana', 983.0, 422.0, 1060665],
[27, 'West North Central', 'Nebraska', 1745.0, 676.0, 1925614, 27,
 'West North Central', 'Nebraska', 1745.0, 676.0, 1925614],
[28, 'Mountain', 'Nevada', 7058.0, 486.0, 3027341, 28, 'Mountain',
```

5. Perform file input and output operations on nd array using save, load, savetxt and loadtxt methods.

The ndarray objects can be saved to and loaded from the disk files. The IO functions available are −

load() and save() functions handle /numPy binary files (with npy extension)

loadtxt() and savetxt() functions handle normal text files

NumPy introduces a simple file format for ndarray objects. This .npy file stores data, shape, dtype and other information required to reconstruct the ndarray in a disk file such that the array is correctly retrieved even if the file is on another machine with different architecture.

numpy.save() The numpy.save() file stores the input array in a disk file with npy extension.

```
import numpy as np
arr = np.array([[7,8,9], [4,5,6]])
```

```
#numpy.save() to save the array to a file
np.save('array.npy',arr)
arr
```

```
array([[7, 8, 9],
       [4, 5, 6]])
```

```
# np.load() use to load the save array from file
load_arr= np.load('array.npy')
print(load_arr)
```

```
[[7 8 9]
 [4 5 6]]
```

```
np.savetxt('array.txt',arr,delimiter=',')
```

```
# Using numpy.loadtxt() to load the saved array from a text file
loaded_arr_txt = np.loadtxt('array.txt', delimiter=',')
print(loaded_arr_txt)
```

```
[[7. 8. 9.]
 [4. 5. 6.]]
```