```python
import numpy as np

# Step (a)
var1 = np.arange(31)
print("var1:", var1)
print("Shape of var1:", var1.shape)

# Step (b)
var2 = var1[1:].reshape(5, 6)
print("\nvar2 (2D matrix):", var2)
print("Shape of var2:", var2.shape)

# Step (c)
var3 = var2.reshape(2, 3, 5)
print("\nvar3 (3D matrix):", var3)
print("Shape of var3:", var3.shape)

# Step (d)
var2[1, 0] = -1
print("\nModified var2:", var2)
print("Modified var1:", var1)
print("Modified var3:", var3)

# Step (i)
sum_var3 = np.sum(var3, axis=1)
print("\nSum of var3 over its second dimension (axis 1):")
print(sum_var3)

# Step (iii)
# Sum var3 over both its first (axis 0) and third (axis 2) dimensions
sum_var3_first_third = np.sum(var3, axis=(0, 2))
print("\nSum of var3 over both its first and third dimensions (axes 0
and 2):")
print(sum_var3_first_third)


#  Write code to do the following:
# (i) Slice out the second row of var2 and print it.
# (ii) Slice out the last column of var2 using the -1 notation and
print it.
# (iii) Slice out the top right 2 × 2 submatrix of var2 and print it

print(var2[1])
print(var2[:, -1])
print(var2[:2, -2:])
```

```
var1: [ 0  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 28 29 30]
Shape of var1: (31,)
```

```
var2 (2D matrix): [[ 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 28 29 30]]
Shape of var2: (5, 6)

var3 (3D matrix): [[[ 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 28 29 30]]]
Shape of var3: (2, 3, 5)

Modified var2: [[ 1  2  3  4  5  6]
 [-1  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]]
Modified var1: [ 0  1  2  3  4  5  6 -1  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23
 24 25 26 27 28 29 30]
Modified var3: [[[ 1  2  3  4  5]
  [ 6 -1  8  9 10]
  [11 12 13 14 15]]

 [[16 17 18 19 20]
  [21 22 23 24 25]
  [26 27 28 29 30]]]

Sum of var3 over its second dimension (axis 1):
[[18 13 24 27 30]
 [63 66 69 72 75]]

Sum of var3 over both its first and third dimensions (axes 0 and 2):
[105 147 205]
[-1  8  9 10 11 12]
[ 6 12 18 24 30]
[[ 5  6]
 [11 12]]
```

```python
import numpy as np

# Create a vector from 1 to 10
vector = np.arange(10) + 1
print("Vector:", vector)
```

```
# Create a 10 × 10 matrix A where A[i][j] = i + j
row = np.arange(10)
column = row.reshape(10, 1)
arr = row + column
print("\nMatrix A:\n", arr)

# Generate a random dataset of integers
data = np.random.randint(0, 100, size=(50, 5))
print("\nRandom integer dataset:\n", data)

# Calculate the mean and standard deviation of the dataset
mean = np.mean(data, axis=0)
std = np.std(data, axis=0)
print("\nMean of the dataset:", mean)
print("Standard deviation of the dataset:", std)

# Normalize the data
normalized = (data - mean) / std

# Convert normalized values to integers (using np.round())
normalized_int = np.round(normalized).astype(int)  # Rounding and
converting to int
print("\nNormalized integer dataset:\n", normalized_int)

# Calculate and print the mean and standard deviation of the
normalized data
mean_normalized = np.mean(normalized_int, axis=0).round()
std_normalized = np.std(normalized_int, axis=0).round()
print("\nMean of normalized integer data:", mean_normalized)
print("Standard deviation of normalized integer data:",
std_normalized)
```

```
Vector: [ 1  2  3  4  5  6  7  8  9 10]

Matrix A:
 [[ 0  1  2  3  4  5  6  7  8  9]
 [ 1  2  3  4  5  6  7  8  9 10]
 [ 2  3  4  5  6  7  8  9 10 11]
 [ 3  4  5  6  7  8  9 10 11 12]
 [ 4  5  6  7  8  9 10 11 12 13]
 [ 5  6  7  8  9 10 11 12 13 14]
 [ 6  7  8  9 10 11 12 13 14 15]
 [ 7  8  9 10 11 12 13 14 15 16]
 [ 8  9 10 11 12 13 14 15 16 17]
 [ 9 10 11 12 13 14 15 16 17 18]]

Random integer dataset:
 [[11 31 85  2 98]
 [42 94 15 92 36]
 [15 83  3 36 47]
 [52 41 76 85 62]
 [61 23 93  0 29]
 [47 23 56 63 11]
 [99 50 41 71 96]
 [ 8 95  2 83 57]
 [ 0 39 49 94 27]
 [35 64 73 80 26]
 [45 29 78 87 72]
 [33 29 76 78  5]
 [84 86 53 14 65]
 [76 59 18 70 85]
 [75 66 90 13  8]
 [70 83 73 70 42]
 [15 72 63 51 51]
 [29 49  1 73 57]
 [63 52 47 86 31]
 [75 14 54 48 20]
 [34  0 28 85 66]
 [83 16 30 35 32]
 [ 2 70 37 86 81]
 [57 44 15 92 94]
 [99 96 71  6 22]
 [44 60 20 46 68]
 [34 35 95 47 65]
 [51 35 64 91 10]
 [71  3 11 39 36]
 [44 62 53 78 35]
 [71 46 62 46 99]
 [17 61 11 39 60]
 [20 93 76 66 12]
 [12 57 35 97 13]
 [30 86 89 60  4]
 [68 49 91  0 89]
 [77 94 12 50 67]
 [92 65 67 71 37]
 [92 97 44 55  3]
 [95 87 24 57  4]
 [45 42 99 54 13]
 [56 55 90 95 60]
 [26 34 44 43 47]
 [96  8 63  5 57]
 [13 21 73 64 94]
 [74 61 27 85 18]
 [93 96 12 47 75]
 [73 44 41 84  1]
 [42 90 48 21  8]
 [96 98 34 48 53]]

Mean of the dataset: [52.84 55.74 50.24 57.76 44.96]
```

```
Standard deviation of the dataset: [29.00576494 27.68307064
28.27759537 28.02895646 29.51742536]

Normalized integer dataset:
 [[-1 -1  1 -2  2]
 [ 0  1 -1  1  0]
 [-1  1 -2 -1  0]
 [ 0 -1  1  1  1]
 [ 0 -1  2 -2 -1]
 [ 0 -1  0  0 -1]
 [ 2  0  0  0  2]
 [-2  1 -2  1  0]
 [-2 -1  0  1 -1]
 [-1  0  1  1 -1]
 [ 0 -1  1  1  1]
 [-1 -1  1  1 -1]
 [ 1  1  0 -2  1]
 [ 1  0 -1  0  1]
 [ 1  0  1 -2 -1]
 [ 1  1  1  0  0]
 [-1  1  0  0  0]
 [-1  0 -2  1  0]
 [ 0  0  0  1  0]
 [ 1 -2  0  0 -1]
 [-1 -2 -1  1  1]
 [ 1 -1 -1 -1  0]
 [-2  1  0  1  1]
 [ 0  0 -1  1  2]
 [ 2  1  1 -2 -1]
 [ 0  0 -1  0  1]
 [-1 -1  2  0  1]
 [ 0 -1  0  1 -1]
 [ 1 -2 -1 -1  0]
 [ 0  0  0  1  0]
 [ 1  0  0  0  2]
 [-1  0 -1 -1  1]
 [-1  1  1  0 -1]
 [-1  0 -1  1 -1]
 [-1  1  1  0 -1]
 [ 1  0  1 -2  1]
 [ 1  1 -1  0  1]
 [ 1  0  1  0  0]
 [ 1  1  0  0 -1]
 [ 1  1 -1  0 -1]
 [ 0  0  2  0 -1]
 [ 0  0  1  1  1]
 [-1 -1  0 -1  0]
 [ 1 -2  0 -2  0]
 [-1 -1  1  0  2]
 [ 1  0 -1  1 -1]
 [ 1  1 -1  0  1]
 [ 1  0  0  1 -1]
 [ 0  1  0 -1 -1]
 [ 1  2 -1  0  0]]

Mean of normalized integer data: [ 0. -0.  0. -0.  0.]
Standard deviation of normalized integer data: [1. 1. 1. 1. 1.]
```

```
import numpy as np

def Vandermonde(N):
    base = np.arange(N, dtype=np.int64) + 1  # Create base array [1,
2, ..., N]
    power = np.arange(N, dtype=np.int64)      # Create power array [0,
1, 2, ..., N-1]
    base = base.reshape(N, 1)                 # Reshape base to (N,
1)

    vander = base ** power                    # Create Vandermonde
matrix

    return vander                             # Return the matrix

# Create Vandermonde matrix for N = 12
vander_matrix = Vandermonde(12)
print(vander_matrix)
# Create a vector of ones of length 12
x = np.ones(12, dtype=np.int64)  # Using int64 for consistency
print(x)
# Perform matrix-vector multiplication
b = vander_matrix @ x  # or you can use np.dot(vander_matrix, x)

# Print the resulting vector b
print("Vector b (result of matrix-vector multiplication):\n", b)

import numpy.linalg as linalg

# Solve for x by inverting the Vandermonde matrix and multiplying by b
# x_solved = linalg.inv(vander_matrix) @ b
x_solved = linalg.solve(vander_matrix, b)

# Print out the result
print("Solved vector x:\n", x_solved)
```

```
[[            1            1            1            1            1
1            1
             1            1            1            1            1]
 [            1            2            4            8           16
32           64
```

```
           128          256          512         1024         2048]
 [           1            3            9           27           81
243          729
          2187         6561        19683        59049       177147]
 [           1            4           16           64          256
1024         4096
         16384        65536       262144      1048576      4194304]
 [           1            5           25          125          625
3125        15625
         78125       390625      1953125      9765625     48828125]
 [           1            6           36          216         1296
7776        46656
        279936      1679616     10077696     60466176    362797056]
 [           1            7           49          343         2401
16807       117649
        823543      5764801     40353607    282475249   1977326743]
 [           1            8           64          512         4096
32768       262144
       2097152     16777216    134217728   1073741824   8589934592]
 [           1            9           81          729         6561
59049       531441
       4782969     43046721    387420489   3486784401  31381059609]
 [           1           10          100         1000        10000
100000      1000000
      10000000    100000000   1000000000  10000000000 100000000000]
 [           1           11          121         1331        14641
161051      1771561
      19487171    214358881   2357947691  25937424601 285311670611]
 [           1           12          144         1728        20736
248832      2985984
      35831808    429981696   5159780352  61917364224 743008370688]]
[1 1 1 1 1 1 1 1 1 1 1 1]
Vector b (result of matrix-vector multiplication):
 [          12         4095       265720      5592405     61035156
435356467   2306881200
   9817068105  35303692060 111111111111 313842837672 810554586205]
Solved vector x:
 [1.07 0.79 1.25 0.83 1.07 0.98 1.   1.   1.   1.   1.   1.  ]
```