NumPy Questions

Question 1: Array Creation and Manipulation

1. Create a NumPy array of shape (5, 5) filled with random integers between 1 and 20. Replace all the elements in the third column with 1.

2. Create a NumPy array of shape (4, 4) with values from 1 to 16. Replace the diagonal elements with 0.

In [1]:
```
import numpy as np

# Create a NumPy array of shape (5, 5) filled with random integers

array = np.random.randint(1, 21, size=(5, 5))
print("Original array:")
print(array)

# Replace all the elements in the third column with 1

array[:, 2] = 1
print("Modified array:")
print(array)
```

```
Original array:
[[19  4  9 13  3]
 [20  4 13 17 14]
 [10  2  4 20  5]
 [18  1 15  9  8]
 [12  7  8 20  8]]
Modified array:
[[19  4  1 13  3]
 [20  4  1 17 14]
 [10  2  1 20  5]
 [18  1  1  9  8]
 [12  7  1 20  8]]
```

In [2]:
```
# Create a NumPy array of shape (4, 4) with values from 1 to 16
array = np.arange(1, 17).reshape((4, 4))
print("Original array:")
```

```
print(array)

# Replace the diagonal elements with 0
np.fill_diagonal(array, 0)
print("Modified array:")
print(array)
```

```
Original array:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Modified array:
[[ 0  2  3  4]
 [ 5  0  7  8]
 [ 9 10  0 12]
 [13 14 15  0]]
```

Question 2: Array Indexing and Slicing

1. Create a NumPy array of shape (6, 6) with values from 1 to 36. Extract the sub-array consisting of the 3rd to 5th rows and 2nd to 4th columns.

2. Create a NumPy array of shape (5, 5) with random integers. Extract the elements on the border.

In [3]:
```
# Create a NumPy array of shape (6, 6) with values from 1 to 36
array = np.arange(1, 37).reshape((6, 6))
print("Original array:")
print(array)

# Extract the sub-array
sub_array = array[2:5, 1:4]
print("Sub-array:")
print(sub_array)
```

```
Original 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 28 29 30]
 [31 32 33 34 35 36]]
Sub-array:
[[14 15 16]
 [20 21 22]
 [26 27 28]]
```

In [4]:
```python
# Create a NumPy array of shape (5, 5) with random integers
array = np.random.randint(1, 21, size=(5, 5))
print("Original array:")
print(array)

# Extract the elements on the border
border_elements = np.concatenate((array[0, :], array[-1, :], array[1:-1, 0], array[1:-1, -1]))
print("Border elements:")
print(border_elements)
```

```
Original array:
[[16 10  1 20  8]
 [11 17 20 17 14]
 [ 9 19 14 20  2]
 [11 15 17  9  8]
 [16  6 13 14  9]]
Border elements:
[16 10  1 20  8 16  6 13 14  9 11  9 11 14  2  8]
```

Question 3: Array Operations

1. Create two NumPy arrays of shape (3, 4) filled with random integers. Perform element-wise addition, subtraction, multiplication, and division.

2. Create a NumPy array of shape (4, 4) with values from 1 to 16. Compute the row-wise and column-wise sum.

In [5]:
```python
# Create two NumPy arrays of shape (3, 4) filled with random integers
array1 = np.random.randint(1, 11, size=(3, 4))
array2 = np.random.randint(1, 11, size=(3, 4))
print("Array 1:")
print(array1)
```

```
print("Array 2:")
print(array2)

# Perform element-wise operations
addition = array1 + array2
subtraction = array1 - array2
multiplication = array1 * array2
division = array1 / array2

print("Element-wise addition:")
print(addition)
print("Element-wise subtraction:")
print(subtraction)
print("Element-wise multiplication:")
print(multiplication)
print("Element-wise division:")
print(division)
```

```
Array 1:
[[ 5  2  3  7]
 [ 6  7  8  2]
 [ 2  1 10  3]]
Array 2:
[[ 8  2  4  3]
 [ 4 10  6  6]
 [ 5  3  7  9]]
Element-wise addition:
[[13  4  7 10]
 [10 17 14  8]
 [ 7  4 17 12]]
Element-wise subtraction:
[[-3  0 -1  4]
 [ 2 -3  2 -4]
 [-3 -2  3 -6]]
Element-wise multiplication:
[[40  4 12 21]
 [24 70 48 12]
 [10  3 70 27]]
Element-wise division:
[[0.625      1.         0.75       2.33333333]
 [1.5        0.7        1.33333333 0.33333333]
 [0.4        0.33333333 1.42857143 0.33333333]]
```

In [6]:
```
# Create a NumPy array of shape (4, 4) with values from 1 to 16
array = np.arange(1, 17).reshape((4, 4))
print("Original array:")
print(array)

# Compute the row-wise and column-wise sum
row_sum = np.sum(array, axis=1)
column_sum = np.sum(array, axis=0)

print("Row-wise sum:")
print(row_sum)
print("Column-wise sum:")
print(column_sum)
```

```
Original array:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Row-wise sum:
[10 26 42 58]
Column-wise sum:
[28 32 36 40]
```

Question 4: Statistical Operations

1. Create a NumPy array of shape (5, 5) filled with random integers. Compute the mean, median, standard deviation, and variance of the array.

2. Create a NumPy array of shape (3, 3) with values from 1 to 9. Normalize the array (i.e., scale the values to have a mean of 0 and a standard deviation of 1).

In [7]:
```
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print("Original array:")
print(array)

# Compute the statistical values
mean = np.mean(array)
median = np.median(array)
std_dev = np.std(array)
variance = np.var(array)
```

```
print("Mean:", mean)
print("Median:", median)
print("Standard Deviation:", std_dev)
print("Variance:", variance)
```

```
Original array:
[[ 7 12 20  8  7]
 [ 8 20 19  4  8]
 [19  5 19  1 15]
 [17 17 11 10  3]
 [16 12  5 18 17]]
Mean: 11.92
Median: 12.0
Standard Deviation: 5.925672957563554
Variance: 35.1136
```

In [8]:
```
# Create a NumPy array of shape (3, 3) with values from 1 to 9
array = np.arange(1, 10).reshape((3, 3))
print("Original array:")
print(array)

# Normalize the array
mean = np.mean(array)
std_dev = np.std(array)
normalized_array = (array - mean) / std_dev

print("Normalized array:")
print(normalized_array)
```

```
Original array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Normalized array:
[[-1.54919334 -1.161895   -0.77459667]
 [-0.38729833  0.          0.38729833]
 [ 0.77459667  1.161895    1.54919334]]
```

Question 5: Broadcasting

1. Create a NumPy array of shape (3, 3) filled with random integers. Add a 1D array of shape (3,) to each row of the 2D array using broadcasting.

2. Create a NumPy array of shape (4, 4) filled with random integers. Subtract a 1D array of shape (4,) from each column of the 2D array using broadcasting.

In [9]:
```
# Create a NumPy array of shape (3, 3) filled with random integers
array = np.random.randint(1, 11, size=(3, 3))
row_array = np.random.randint(1, 11, size=(3,))
print("Original array:")
print(array)
print("1D array:")
print(row_array)

# Add the 1D array to each row of the 2D array using broadcasting
result = array + row_array
print("Resulting array:")
print(result)
```

```
Original array:
[[ 8 10  7]
 [ 1 10  5]
 [ 9  4  1]]
1D array:
[2 6 8]
Resulting array:
[[10 16 15]
 [ 3 16 13]
 [11 10  9]]
```

In [10]:
```
# Create a NumPy array of shape (4, 4) filled with random integers
array = np.random.randint(1, 11, size=(4, 4))
column_array = np.random.randint(1, 11, size=(4,))
print("Original array:")
print(array)
print("1D array:")
print(column_array)

# Subtract the 1D array from each column of the 2D array using broadcasting
result = array - column_array[:, np.newaxis]
print("Resulting array:")
print(result)
```

```
Original array:
[[8 5 3 9]
 [9 9 3 3]
 [6 4 1 7]
 [2 7 4 6]]
1D array:
[7 2 4 3]
Resulting array:
[[ 1 -2 -4  2]
 [ 7  7  1  1]
 [ 2  0 -3  3]
 [-1  4  1  3]]
```

Question 6: Linear Algebra

1. Create a NumPy array of shape (3, 3) representing a matrix. Compute its determinant, inverse, and eigenvalues.

2. Create two NumPy arrays of shape (2, 3) and (3, 2). Perform matrix multiplication on these arrays.

In [11]:
```python
# Create a NumPy array of shape (3, 3) representing a matrix
matrix = np.random.randint(1, 11, size=(3, 3))
print("Original matrix:")
print(matrix)

# Compute the determinant
determinant = np.linalg.det(matrix)
print("Determinant:", determinant)

# Compute the inverse
inverse = np.linalg.inv(matrix)
print("Inverse:")
print(inverse)

# Compute the eigenvalues
eigenvalues = np.linalg.eigvals(matrix)
print("Eigenvalues:", eigenvalues)
```

```
Original matrix:
[[10  1  3]
 [ 1  9  4]
 [ 4  9  6]]
Determinant: 108.99999999999997
Inverse:
[[ 0.16513761  0.19266055 -0.21100917]
 [ 0.09174312  0.44036697 -0.33944954]
 [-0.24770642 -0.78899083  0.81651376]]
Eigenvalues: [15.38686093  8.80896191  0.80417717]
```

In [12]:
```python
# Create two NumPy arrays of shape (2, 3) and (3, 2)
array1 = np.random.randint(1, 11, size=(2, 3))
array2 = np.random.randint(1, 11, size=(3, 2))
print("Array 1:")
print(array1)
print("Array 2:")
print(array2)

# Perform matrix multiplication
result = np.dot(array1, array2)
print("Matrix multiplication result:")
print(result)
```

```
Array 1:
[[1 9 7]
 [8 4 4]]
Array 2:
[[5 6]
 [2 4]
 [6 3]]
Matrix multiplication result:
[[65 63]
 [72 76]]
```

Question 7: Advanced Array Manipulation

1. Create a NumPy array of shape (3, 3) with values from 1 to 9. Reshape the array to shape (1, 9) and then to shape (9, 1).

2. Create a NumPy array of shape (5, 5) filled with random integers. Flatten the array and then reshape it back to (5, 5).

In [13]:
```
# Create a NumPy array of shape (3, 3) with values from 1 to 9
array = np.arange(1, 10).reshape((3, 3))
print("Original array:")
print(array)

# Reshape the array to shape (1, 9)
reshaped_array_1 = array.reshape((1, 9))
print("Reshaped array (1, 9):")
print(reshaped_array_1)

# Reshape the array to shape (9, 1)
reshaped_array_2 = reshaped_array_1.reshape((9, 1))
print("Reshaped array (9, 1):")
print(reshaped_array_2)
```

```
Original array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Reshaped array (1, 9):
[[1 2 3 4 5 6 7 8 9]]
Reshaped array (9, 1):
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
```

In [14]:
```
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print("Original array:")
print(array)

# Flatten the array
flattened_array = array.flatten()
print("Flattened array:")
print(flattened_array)

# Reshape the array back to (5, 5)
```

```
reshaped_array = flattened_array.reshape((5, 5))
print("Reshaped array:")
print(reshaped_array)
```

```
Original array:
[[17 18 16 17  7]
 [15  7 18  1 14]
 [ 9  6  5  8 11]
 [ 8  2 10  3  5]
 [14 18 13  6 15]]
Flattened array:
[17 18 16 17  7 15  7 18  1 14  9  6  5  8 11  8  2 10  3  5 14 18 13  6
 15]
Reshaped array:
[[17 18 16 17  7]
 [15  7 18  1 14]
 [ 9  6  5  8 11]
 [ 8  2 10  3  5]
 [14 18 13  6 15]]
```

Question 8: Fancy Indexing and Boolean Indexing

1. Create a NumPy array of shape (5, 5) filled with random integers. Use fancy indexing to extract the elements at the corners of the array.

2. Create a NumPy array of shape (4, 4) filled with random integers. Use boolean indexing to set all elements greater than 10 to 10.

In [15]:
```
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print("Original array:")
print(array)

# Use fancy indexing to extract the elements at the corners of the array
corners = array[[0, 0, -1, -1], [0, -1, 0, -1]]
print("Corner elements:")
print(corners)
```

```
Original array:
[[14 18  7 18 16]
 [ 8  6  9 12  1]
 [14 12 17 17 11]
 [ 8 13 20  9 19]
 [12 19 17 11 15]]
Corner elements:
[14 16 12 15]
```

In [16]:
```
# Create a NumPy array of shape (4, 4) filled with random integers
array = np.random.randint(1, 21, size=(4, 4))
print("Original array:")
print(array)

# Use boolean indexing to set all elements greater than 10 to 10
array[array > 10] = 10
print("Modified array:")
print(array)
```

```
Original array:
[[20  7  3  7]
 [16  9  2 13]
 [ 7  9 14  7]
 [ 4  8  3  9]]
Modified array:
[[10  7  3  7]
 [10  9  2 10]
 [ 7  9 10  7]
 [ 4  8  3  9]]
```

Question 9: Structured Arrays

1. Create a structured array with fields 'name' (string), 'age' (integer), and 'weight' (float). Add some data and sort the array by age.

2. Create a structured array with fields 'x' and 'y' (both integers). Add some data and compute the Euclidean distance between each pair of points.

In [17]:
```
# Create a structured array with fields 'name', 'age', and 'weight'
data_type = [('name', 'U10'), ('age', 'i4'), ('weight', 'f4')]
data = np.array([('Alice', 25, 55.5), ('Bob', 30, 85.3), ('Charlie', 20, 65.2)], dtype=data_type)
print("Original array:")
print(data)
```

```
# Sort the array by age
sorted_data = np.sort(data, order='age')
print("Sorted array by age:")
print(sorted_data)
```

```
Original array:
[('Alice', 25, 55.5) ('Bob', 30, 85.3) ('Charlie', 20, 65.2)]
Sorted array by age:
[('Charlie', 20, 65.2) ('Alice', 25, 55.5) ('Bob', 30, 85.3)]
```

In [18]:
```
# Create a structured array with fields 'x' and 'y'
data_type = [('x', 'i4'), ('y', 'i4')]
data = np.array([(1, 2), (3, 4), (5, 6)], dtype=data_type)
print("Original array:")
print(data)

# Compute the Euclidean distance between each pair of points
distances = np.sqrt((data['x'][:, np.newaxis] - data['x'])**2 + (data['y'][:, np.newaxis] - data['y'])**2)
print("Euclidean distances:")
print(distances)
```

```
Original array:
[(1, 2) (3, 4) (5, 6)]
Euclidean distances:
[[0.         2.82842712 5.65685425]
 [2.82842712 0.         2.82842712]
 [5.65685425 2.82842712 0.        ]]
```

Question 10: Masked Arrays

1. Create a masked array of shape (4, 4) with random integers and mask the elements greater than 10. Compute the sum of the unmasked elements.

2. Create a masked array of shape (3, 3) with random integers and mask the diagonal elements. Replace the masked elements with the mean of the unmasked elements.

In [19]:
```
import numpy.ma as ma

# Create a masked array of shape (4, 4) with random integers
array = np.random.randint(1, 21, size=(4, 4))
masked_array = ma.masked_greater(array, 10)
```

```
print("Original array:")
print(array)
print("Masked array:")
print(masked_array)

# Compute the sum of the unmasked elements
sum_unmasked = masked_array.sum()
print("Sum of unmasked elements:", sum_unmasked)
```

```
Original array:
[[20 16  2  7]
 [20 19 19 19]
 [ 2 17  1 12]
 [ 5  5 10 18]]
Masked array:
[[-- -- 2 7]
 [-- -- -- --]
 [2 -- 1 --]
 [5 5 10 --]]
Sum of unmasked elements: 32
```

In [20]:

```
# Create a masked array of shape (3, 3) with random integers
array = np.random.randint(1, 21, size=(3, 3))
masked_array = ma.masked_array(array, mask=np.eye(3, dtype=bool))
print("Original array:")
print(array)
print("Masked array:")
print(masked_array)

# Replace the masked elements with the mean of the unmasked elements
mean_unmasked = masked_array.mean()
masked_array = masked_array.filled(mean_unmasked)
print("Modified masked array:")
print(masked_array)
```

```
Original array:
[[ 2 10  8]
 [ 8 10  4]
 [19 18 18]]
Masked array:
[[-- 10 8]
 [8 -- 4]
 [19 18 --]]
Modified masked array:
[[11 10  8]
 [ 8 11  4]
 [19 18 11]]
```

Pandas Questions

Assignment 1: DataFrame Creation and Indexing

1. Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers. Set the index to be the first column.

2. Create a Pandas DataFrame with columns 'A', 'B', 'C' and index 'X', 'Y', 'Z'. Fill the DataFrame with random integers and access the element at row 'Y' and column 'B'.

In [21]:
```python
import pandas as pd
import numpy as np

# Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 4)), columns=['A', 'B', 'C', 'D'])
print("Original DataFrame:")
print(df)

# Set the index to be the first column
df.set_index('A', inplace=True)
print("DataFrame with new index:")
print(df)
```

```
Original DataFrame:
    A   B   C   D
0  57  82  97  53
1   6   9  86  85
2  95  78  75  48
3  44  83  36  71
4  82  76  20  73
5  74  96  87  84
DataFrame with new index:
     B   C   D
A
57  82  97  53
6    9  86  85
95  78  75  48
44  83  36  71
82  76  20  73
74  96  87  84
```

In [22]:
```python
# Create a Pandas DataFrame with specified columns and index
df = pd.DataFrame(np.random.randint(1, 100, size=(3, 3)), columns=['A', 'B', 'C'], index=['X', 'Y', 'Z'])
print("Original DataFrame:")
print(df)

# Access the element at row 'Y' and column 'B'
element = df.at['Y', 'B']
print("Element at row 'Y' and column 'B':", element)
```

```
Original DataFrame:
    A   B   C
X  89  68  42
Y   4  58   2
Z  20  64  48
Element at row 'Y' and column 'B': 58
```

Assignment 2: DataFrame Operations

1. Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers. Add a new column that is the product of the first two columns.

2. Create a Pandas DataFrame with 3 columns and 4 rows filled with random integers. Compute the row-wise and column-wise sum.

In [23]:
```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
print("Original DataFrame:")
print(df)

# Add a new column that is the product of the first two columns
df['D'] = df['A'] * df['B']
print("DataFrame with new column:")
print(df)
```

```
Original DataFrame:
    A   B   C
0  91  10  43
1  64  42  35
2  76  15  86
3  41  43   1
4   3  18  81
DataFrame with new column:
    A   B   C     D
0  91  10  43   910
1  64  42  35  2688
2  76  15  86  1140
3  41  43   1  1763
4   3  18  81    54
```

In [24]:
```
# Create a Pandas DataFrame with 3 columns and 4 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(4, 3)), columns=['A', 'B', 'C'])
print("Original DataFrame:")
print(df)

# Compute the row-wise and column-wise sum
row_sum = df.sum(axis=1)
column_sum = df.sum(axis=0)

print("Row-wise sum:")
print(row_sum)
print("Column-wise sum:")
print(column_sum)
```

```
Original DataFrame:
    A   B   C
0  95  91  45
1  80  46  35
2  52  46  41
3  69  43  84
Row-wise sum:
0    231
1    161
2    139
3    196
dtype: int64
Column-wise sum:
A    296
B    226
C    205
dtype: int64
```

Assignment 3: Data Cleaning

1. Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers. Introduce some NaN values. Fill the NaN values with the mean of the respective columns.

2. Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers. Introduce some NaN values. Drop the rows with any NaN values.

In [25]:
```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
print("Original DataFrame:")
print(df)

# Introduce some NaN values
df.iloc[0, 1] = np.nan
df.iloc[2, 2] = np.nan
df.iloc[4, 0] = np.nan
print("DataFrame with NaN values:")
print(df)

# Fill the NaN values with the mean of the respective columns
df.fillna(df.mean(), inplace=True)
```

```
print("DataFrame with NaN values filled:")
print(df)
```

```
Original DataFrame:
    A   B   C
0  98   1  76
1  84  63  87
2  74  23  14
3   7  30  62
4  61  53  51
DataFrame with NaN values:
      A     B     C
0  98.0   NaN  76.0
1  84.0  63.0  87.0
2  74.0  23.0   NaN
3   7.0  30.0  62.0
4   NaN  53.0  51.0
DataFrame with NaN values filled:
       A      B     C
0  98.00  42.25  76.0
1  84.00  63.00  87.0
2  74.00  23.00  69.0
3   7.00  30.00  62.0
4  65.75  53.00  51.0
```

In [26]:
```
# Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 4)), columns=['A', 'B', 'C', 'D'])
print("Original DataFrame:")
print(df)

# Introduce some NaN values
df.iloc[1, 2] = np.nan
df.iloc[3, 0] = np.nan
df.iloc[5, 1] = np.nan
print("DataFrame with NaN values:")
print(df)

# Drop the rows with any NaN values
df.dropna(inplace=True)
print("DataFrame with NaN values dropped:")
print(df)
```

```
Original DataFrame:
    A   B   C   D
0  95  98  92   6
1   5   6  94   2
2  73  33  63  15
3  54  87  19  17
4  52  36  79  76
5  78  31  23  97
DataFrame with NaN values:
      A     B     C   D
0  95.0  98.0  92.0   6
1   5.0   6.0   NaN   2
2  73.0  33.0  63.0  15
3   NaN  87.0  19.0  17
4  52.0  36.0  79.0  76
5  78.0   NaN  23.0  97
DataFrame with NaN values dropped:
      A     B     C   D
0  95.0  98.0  92.0   6
2  73.0  33.0  63.0  15
4  52.0  36.0  79.0  76
```

Assignment 4: Data Aggregation

1. Create a Pandas DataFrame with 2 columns: 'Category' and 'Value'. Fill the 'Category' column with random categories ('A', 'B', 'C') and the 'Value' column with random integers. Group the DataFrame by 'Category' and compute the sum and mean of 'Value' for each category.

2. Create a Pandas DataFrame with 3 columns: 'Product', 'Category', and 'Sales'. Fill the DataFrame with random data. Group the DataFrame by 'Category' and compute the total sales for each category.

In [27]:
```
# Create a Pandas DataFrame with 2 columns: 'Category' and 'Value'
df = pd.DataFrame({'Category': np.random.choice(['A', 'B', 'C'], size=10), 'Value': np.random.randint(1, 100, size=10)})
print("Original DataFrame:")
print(df)

# Group the DataFrame by 'Category' and compute the sum and mean of 'Value' for each category
grouped = df.groupby('Category')['Value'].agg(['sum', 'mean'])
print("Grouped DataFrame:")
print(grouped)
```

```
Original DataFrame:
  Category  Value
0        A     10
1        C     24
2        B     71
3        C     87
4        B     84
5        B     89
6        A     56
7        C     23
8        A     26
9        C     82
Grouped DataFrame:
           sum       mean
Category
A           92  30.666667
B          244  81.333333
C          216  54.000000
```

In [28]:

```python
# Create a Pandas DataFrame with 3 columns: 'Product', 'Category', and 'Sales'
df = pd.DataFrame({'Product': np.random.choice(['Prod1', 'Prod2', 'Prod3'], size=10), 'Category': np.random.choice(['A', 'B', 'C'
print("Original DataFrame:")
print(df)

# Group the DataFrame by 'Category' and compute the total sales for each category
grouped = df.groupby('Category')['Sales'].sum()
print("Grouped DataFrame:")
print(grouped)
```

```
Original DataFrame:
  Product Category  Sales
0   Prod1        C     30
1   Prod1        B     23
2   Prod1        B     13
3   Prod1        C     23
4   Prod1        C     75
5   Prod2        B     17
6   Prod1        C     55
7   Prod3        A     26
8   Prod2        A     89
9   Prod2        A     10
Grouped DataFrame:
Category
A    125
B     53
C    183
Name: Sales, dtype: int64
```

Assignment 5: Merging DataFrames

1. Create two Pandas DataFrames with a common column. Merge the DataFrames using the common column.

2. Create two Pandas DataFrames with different columns. Concatenate the DataFrames along the rows and along the columns.

In [29]:
```python
# Create two Pandas DataFrames with a common column
df1 = pd.DataFrame({'Key': ['A', 'B', 'C', 'D'], 'Value1': np.random.randint(1, 100, size=4)})
df2 = pd.DataFrame({'Key': ['A', 'B', 'C', 'E'], 'Value2': np.random.randint(1, 100, size=4)})
print("DataFrame 1:")
print(df1)
print("DataFrame 2:")
print(df2)

# Merge the DataFrames using the common column
merged = pd.merge(df1, df2, on='Key')
print("Merged DataFrame:")
print(merged)
```

```
DataFrame 1:
  Key  Value1
0   A      20
1   B      23
2   C      58
3   D      95
DataFrame 2:
  Key  Value2
0   A      60
1   B      86
2   C       2
3   E      93
Merged DataFrame:
  Key  Value1  Value2
0   A      20      60
1   B      23      86
2   C      58       2
```

In [30]:

```python
# Create two Pandas DataFrames with different columns
df1 = pd.DataFrame({'A': np.random.randint(1, 100, size=3), 'B': np.random.randint(1, 100, size=3)})
df2 = pd.DataFrame({'C': np.random.randint(1, 100, size=3), 'D': np.random.randint(1, 100, size=3)})
print("DataFrame 1:")
print(df1)
print("DataFrame 2:")
print(df2)

# Concatenate the DataFrames along the rows
concat_rows = pd.concat([df1, df2], axis=0)
print("Concatenated DataFrame (rows):")
print(concat_rows)

# Concatenate the DataFrames along the columns
concat_columns = pd.concat([df1, df2], axis=1)
print("Concatenated DataFrame (columns):")
print(concat_columns)
```

```
DataFrame 1:
    A   B
0  13  35
1   1  35
2  18  23
DataFrame 2:
    C   D
0  21  75
1  43  60
2  39  78
Concatenated DataFrame (rows):
      A     B     C     D
0  13.0  35.0   NaN   NaN
1   1.0  35.0   NaN   NaN
2  18.0  23.0   NaN   NaN
0   NaN   NaN  21.0  75.0
1   NaN   NaN  43.0  60.0
2   NaN   NaN  39.0  78.0
Concatenated DataFrame (columns):
    A   B   C   D
0  13  35  21  75
1   1  35  43  60
2  18  23  39  78
```

Assignment 6: Time Series Analysis

1. Create a Pandas DataFrame with a datetime index and one column filled with random integers. Resample the DataFrame to compute the monthly mean of the values.

2. Create a Pandas DataFrame with a datetime index ranging from '2021-01-01' to '2021-12-31' and one column filled with random integers. Compute the rolling mean with a window of 7 days.

In [31]:
```
# Create a Pandas DataFrame with a datetime index and one column filled with random integers
date_rng = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
df = pd.DataFrame(date_rng, columns=['date'])
df['data'] = np.random.randint(0, 100, size=(len(date_rng)))
df.set_index('date', inplace=True)
print("Original DataFrame:")
print(df)

# Resample the DataFrame to compute the monthly mean of the values
```

```
monthly_mean = df.resample('M').mean()
print("Monthly mean DataFrame:")
print(monthly_mean)
```

```
Original DataFrame:
            data
date
2022-01-01    66
2022-01-02    27
2022-01-03    64
2022-01-04    78
2022-01-05    48
...          ...
2022-12-27    27
2022-12-28    91
2022-12-29    34
2022-12-30    31
2022-12-31    30

[365 rows x 1 columns]
Monthly mean DataFrame:
                data
date
2022-01-31  51.258065
2022-02-28  44.178571
2022-03-31  53.677419
2022-04-30  36.200000
2022-05-31  49.709677
2022-06-30  47.000000
2022-07-31  51.612903
2022-08-31  51.451613
2022-09-30  51.500000
2022-10-31  40.967742
2022-11-30  42.366667
2022-12-31  52.225806
```

```
<ipython-input-31-5fb4668014f5>:10: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' in
stead.
  monthly_mean = df.resample('M').mean()
```

In [32]:
```
# Create a Pandas DataFrame with a datetime index ranging from '2021-01-01' to '2021-12-31'
date_rng = pd.date_range(start='2021-01-01', end='2021-12-31', freq='D')
df = pd.DataFrame(date_rng, columns=['date'])
df['data'] = np.random.randint(0, 100, size=(len(date_rng)))
df.set_index('date', inplace=True)
```

```
print("Original DataFrame:")
print(df)

# Compute the rolling mean with a window of 7 days
rolling_mean = df.rolling(window=7).mean()
print("Rolling mean DataFrame:")
print(rolling_mean)
```

```
Original DataFrame:
            data
date
2021-01-01    36
2021-01-02    31
2021-01-03    39
2021-01-04    50
2021-01-05    68
...          ...
2021-12-27    93
2021-12-28    90
2021-12-29    60
2021-12-30    90
2021-12-31    39

[365 rows x 1 columns]
Rolling mean DataFrame:
                data
date
2021-01-01       NaN
2021-01-02       NaN
2021-01-03       NaN
2021-01-04       NaN
2021-01-05       NaN
...              ...
2021-12-27  67.428571
2021-12-28  66.285714
2021-12-29  69.142857
2021-12-30  69.142857
2021-12-31  74.571429

[365 rows x 1 columns]
```

Assignment 7: MultiIndex DataFrame

1. Create a Pandas DataFrame with a MultiIndex (hierarchical index). Perform some basic indexing and slicing operations on the MultiIndex DataFrame.

2. Create a Pandas DataFrame with MultiIndex consisting of 'Category' and 'SubCategory'. Fill the DataFrame with random data and compute the sum of values for each 'Category' and 'SubCategory'.

In [33]:
```
# Create a Pandas DataFrame with a MultiIndex (hierarchical index)
arrays = [['A', 'A', 'B', 'B'], ['one', 'two', 'one', 'two']]
index = pd.MultiIndex.from_arrays(arrays, names=('Category', 'SubCategory'))
df = pd.DataFrame(np.random.randint(1, 100, size=(4, 3)), index=index, columns=['Value1', 'Value2', 'Value3'])
print("MultiIndex DataFrame:")
print(df)

# Basic indexing and slicing operations
print("Indexing at Category 'A':")
print(df.loc['A'])

print("Slicing at Category 'B' and SubCategory 'two':")
print(df.loc[('B', 'two')])
```

```
MultiIndex DataFrame:
                     Value1  Value2  Value3
Category SubCategory
A        one             22      83      67
         two             46      15      25
B        one              2      90      49
         two             98      37      97
Indexing at Category 'A':
             Value1  Value2  Value3
SubCategory
one              22      83      67
two              46      15      25
Slicing at Category 'B' and SubCategory 'two':
Value1    98
Value2    37
Value3    97
Name: (B, two), dtype: int64
```

In [34]:
```
# Create a Pandas DataFrame with MultiIndex consisting of 'Category' and 'SubCategory'
arrays = [['A', 'A', 'B', 'B', 'C', 'C'], ['one', 'two', 'one', 'two', 'one', 'two']]
index = pd.MultiIndex.from_arrays(arrays, names=('Category', 'SubCategory'))
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 3)), index=index, columns=['Value1', 'Value2', 'Value3'])
```

```
print("MultiIndex DataFrame:")
print(df)

# Compute the sum of values for each 'Category' and 'SubCategory'
sum_values = df.groupby(['Category', 'SubCategory']).sum()
print("Sum of values:")
print(sum_values)
```

```
MultiIndex DataFrame:
                     Value1  Value2  Value3
Category SubCategory
A        one             99      82      49
         two             50      44      69
B        one             76      70      65
         two             40      54      53
C        one             71      52      41
         two             64      15      62
Sum of values:
                     Value1  Value2  Value3
Category SubCategory
A        one             99      82      49
         two             50      44      69
B        one             76      70      65
         two             40      54      53
C        one             71      52      41
         two             64      15      62
```

Assignment 8: Pivot Tables

1. Create a Pandas DataFrame with columns 'Date', 'Category', and 'Value'. Create a pivot table to compute the sum of 'Value' for each 'Category' by 'Date'.

2. Create a Pandas DataFrame with columns 'Year', 'Quarter', and 'Revenue'. Create a pivot table to compute the mean 'Revenue' for each 'Quarter' by 'Year'.

In [35]:
```
# Create a Pandas DataFrame with columns 'Date', 'Category', and 'Value'
date_rng = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D')
df = pd.DataFrame({'Date': np.random.choice(date_rng, size=20), 'Category': np.random.choice(['A', 'B', 'C'], size=20), 'Value':
print("Original DataFrame:")
print(df)
```

```
# Create a pivot table to compute the sum of 'Value' for each 'Category' by 'Date'
pivot_table = df.pivot_table(values='Value', index='Date', columns='Category', aggfunc='sum')
print("Pivot Table:")
print(pivot_table)
```

```
Original DataFrame:
         Date Category  Value
0  2022-01-04        B     27
1  2022-01-04        C     22
2  2022-01-04        B      2
3  2022-01-09        A     21
4  2022-01-10        A     88
5  2022-01-07        A     80
6  2022-01-10        A     68
7  2022-01-04        B     44
8  2022-01-06        B     64
9  2022-01-01        B     75
10 2022-01-07        A     70
11 2022-01-06        C     65
12 2022-01-06        B     25
13 2022-01-02        A     33
14 2022-01-08        B     33
15 2022-01-02        C     14
16 2022-01-02        A     16
17 2022-01-06        A     25
18 2022-01-03        B     51
19 2022-01-07        C     18
Pivot Table:
Category        A     B     C
Date
2022-01-01    NaN  75.0   NaN
2022-01-02   49.0   NaN  14.0
2022-01-03    NaN  51.0   NaN
2022-01-04    NaN  73.0  22.0
2022-01-06   25.0  89.0  65.0
2022-01-07  150.0   NaN  18.0
2022-01-08    NaN  33.0   NaN
2022-01-09   21.0   NaN   NaN
2022-01-10  156.0   NaN   NaN
```

In [36]:

```
# Create a Pandas DataFrame with columns 'Year', 'Quarter', and 'Revenue'
df = pd.DataFrame({'Year': np.random.choice([2020, 2021, 2022], size=12), 'Quarter': np.random.choice(['Q1', 'Q2', 'Q3', 'Q4'], s
print("Original DataFrame:")
print(df)
```

```
# Create a pivot table to compute the mean 'Revenue' for each 'Quarter' by 'Year'
pivot_table = df.pivot_table(values='Revenue', index='Year', columns='Quarter', aggfunc='mean')
print("Pivot Table:")
print(pivot_table)
```

```
Original DataFrame:
    Year Quarter  Revenue
0   2021      Q1      462
1   2022      Q3      662
2   2020      Q2      837
3   2021      Q3      802
4   2020      Q3      277
5   2021      Q2      513
6   2022      Q4       56
7   2022      Q2      130
8   2021      Q4      533
9   2022      Q3      930
10  2021      Q2      674
11  2022      Q1      744
Pivot Table:
Quarter     Q1     Q2     Q3     Q4
Year
2020       NaN  837.0  277.0    NaN
2021     462.0  593.5  802.0  533.0
2022     744.0  130.0  796.0   56.0
```

Assignment 9: Applying Functions

1. Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers. Apply a function that doubles the values of the DataFrame.

2. Create a Pandas DataFrame with 3 columns and 6 rows filled with random integers. Apply a lambda function to create a new column that is the sum of the existing columns.

In [37]:
```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
print("Original DataFrame:")
print(df)

# Apply a function that doubles the values of the DataFrame
```

```
df_doubled = df.applymap(lambda x: x * 2)
print("Doubled DataFrame:")
print(df_doubled)
```

```
Original DataFrame:
    A   B   C
0  37   9  75
1  62  42  85
2  55   3  31
3   4  84  89
4  10  20  46
Doubled DataFrame:
     A    B    C
0   74   18  150
1  124   84  170
2  110    6   62
3    8  168  178
4   20   40   92
```

```
<ipython-input-37-79053e5e385f>:7: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  df_doubled = df.applymap(lambda x: x * 2)
```

In [38]:

```
# Create a Pandas DataFrame with 3 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 3)), columns=['A', 'B', 'C'])
print("Original DataFrame:")
print(df)

# Apply a lambda function to create a new column that is the sum of the existing columns
df['Sum'] = df.apply(lambda row: row.sum(), axis=1)
print("DataFrame with Sum column:")
print(df)
```

```
Original DataFrame:
    A   B   C
0  10  53  60
1  29   4  50
2  22  18  86
3  64  92  81
4  74  58  67
5   9  52   7
DataFrame with Sum column:
    A   B   C  Sum
0  10  53  60  123
1  29   4  50   83
2  22  18  86  126
3  64  92  81  237
4  74  58  67  199
5   9  52   7   68
```

Assignment 10: Working with Text Data

1. Create a Pandas Series with 5 random text strings. Convert all the strings to uppercase.

2. Create a Pandas Series with 5 random text strings. Extract the first three characters of each string.

In [39]:
```
# Create a Pandas Series with 5 random text strings
text_data = pd.Series(['apple', 'banana', 'cherry', 'date', 'elderberry'])
print("Original Series:")
print(text_data)

# Convert all the strings to uppercase
uppercase_data = text_data.str.upper()
print("Uppercase Series:")
print(uppercase_data)
```

```
Original Series:
0         apple
1        banana
2        cherry
3          date
4    elderberry
dtype: object
Uppercase Series:
0         APPLE
1        BANANA
2        CHERRY
3          DATE
4    ELDERBERRY
dtype: object
```

In [41]:
```
# Create a Pandas Series with 5 random text strings
text_data = pd.Series(['apple', 'banana', 'cherry', 'date', 'elderberry'])
print("Original Series:")
print(text_data)

# Extract the first three characters of each string
first_three_chars = text_data.str[:3]
print("First three characters:")
print(first_three_chars)
```

```
Original Series:
0         apple
1        banana
2        cherry
3          date
4    elderberry
dtype: object
First three characters:
0    app
1    ban
2    che
3    dat
4    eld
dtype: object
```

In [ ]: