Numpy arrays

In [1]:

```
import numpy as np

#np_heights = np.array([[1.60, 1.75], [1.56, 1.70], [1.49, 1.68]])

np_heights = np.array(
    [[1.60, 1.75],
     [1.56, 1.70],
     [1.49, 1.68]]
)
```

Get array shape

In [32]:

```
print(np_heights.shape)
```

```
(3, 2)
```

The `np_heights.shape` attribute returns the shape of the `np_heights` array, which represents the dimensions of the array. In this case, when you execute `np_heights.shape`, it will return the tuple `(3, 2)`.

The first element of the tuple, `3`, represents the number of rows in the array, indicating that there are three rows. Each row corresponds to the height measurements of a different individual.

The second element of the tuple, `2`, represents the number of columns in the array, indicating that there are two columns. Each column represents a different measurement, specifically the height of each individual in meters.

So, `(3, 2)` indicates that the `np_heights` array has three rows and two columns.

Get the first row of the array

In [36]:

```
print(np_heights[0])
```

```
[1.6  1.75]
```

Get the first value (column) of each row

In [27]:

```
print(np_heights[0:,0])
```

```
[1.6  1.56 1.49]
```

Get the second value (column) of each row

In [28]:

```
print(np_heights[0:,1])
```

```
[1.75 1.7  1.68]
```

In [ ]:

```

```

Coefficient of Correletion

In [33]:

```
print(np.corrcoef(np_heights[0:,0], np_heights[0:,1]))
```

```
[[1.         0.92155064]
 [0.92155064 1.        ]]
```

Consider the follwing numpy array

In [38]:

```
my_array = np.array(
    [[ 5,  1,  6,  2,  4,  6],
     [ 5,  8,  0,  2,  1,  2],
     [ 2,  9,  6, 23, 13,  1]]
)
```

Sum each row of the array

In [50]:

```
my_array.sum(axis=1, keepdims=True)
```

Out[50]:

```
array([[24],
       [18],
       [54]])
```

Sum each column

In [51]:

```
my_array.sum(axis=0, keepdims=True)
```

Out[51]:

```
array([[12, 18, 12, 27, 18,  9]])
```

Take the columns of array and sum this columns and concatenate the sum with the same array to make a new array, such that the fourth row of the new array be formed with the the sum.

In [57]:

```
my_array_sum = my_array.sum(axis=0, keepdims=True)
np.concatenate((my_array, my_array_sum))
```

Out[57]:

```
array([[ 5,  1,  6,  2,  4,  6],
       [ 5,  8,  0,  2,  1,  2],
       [ 2,  9,  6, 23, 13,  1],
       [12, 18, 12, 27, 18,  9]])
```

3D Arrays

In [61]:

```
rgb = np.array(
    [[[255, 0, 0], [0, 0, 0], [0, 0, 255]],
     [[255, 0, 255], [255, 255, 255], [255, 255, 0]]]
)
```

```
print(rgb)
```

```
[[[255   0   0]
  [  0   0   0]
  [  0   0 255]]

 [[255   0 255]
  [255 255 255]
  [255 255   0]]]
```

The code above represents a NumPy array called "rgb" with a shape of (2, 3, 3). This means it is a 3-dimensional array with two layers, each containing 3 rows and 3 columns. Each element of the array represents an RGB color value.

Here's a breakdown of the array:

Layer 1:

- Row 1: [255, 0, 0] (red color)
- Row 2: [0, 0, 0] (black color)
- Row 3: [0, 0, 255] (blue color)

Layer 2:

- Row 1: [255, 0, 255] (purple color)
- Row 2: [255, 255, 255] (white color)
- Row 3: [255, 255, 0] (yellow color)

So, the array "rgb" represents a 2-layer image, where each layer has 3x3 pixels, and each pixel is represented by an RGB color value.

Flip the elements along the specified axis

By default, if no axis is specified, it will flip the array along all axes. Here's the result of applying `np.flip()` to the `rgb` array:

In [72]:

```
print(rgb)
```

```
[[[255   0   0]
  [  0   0   0]
  [  0   0 255]]

 [[255   0 255]
  [255 255 255]
  [255 255   0]]]
```

In [60]:

```
np.flip(rgb)
```

```
Out[60]: array([[[  0, 255, 255],
                 [255, 255, 255],
                 [255,   0, 255]],

                [[255,   0,   0],
                 [  0,   0,   0],
                 [  0,   0, 255]]])
```

The array has been flipped along all axes, so the order of the elements has been reversed. The first layer is now at the bottom, and the second layer is at the top. Additionally, within each layer, the rows and columns have been reversed.

```
In [ ]:
```

If you apply the `np.flip()` function to the `rgb` array with axis=1, it will flip the elements along the specified axis, which in this case is the second axis `(axis=1)`. Here's the result of applying `np.flip(rgb, axis=1)`:

```
In [70]: print(rgb)
```

```
[[[255   0   0]
  [  0   0   0]
  [  0   0 255]]

 [[255   0 255]
  [255 255 255]
  [255 255   0]]]
```

```
In [68]: np.flip(rgb, axis=1)
```

```
Out[68]: array([[[  0,   0, 255],
                 [  0,   0,   0],
                 [255,   0,   0]],

                [[255, 255,   0],
                 [255, 255, 255],
                 [255,   0, 255]]])
```

The elements of the array have been flipped along the second axis, which corresponds to flipping the columns. So, within each layer, the columns have been reversed. The colors are now arranged from right to left within each row.

```
In [ ]:
```

If you apply the `np.flip()` function to the `rgb` array with `axis=2`, it will flip the elements along the specified axis, which in this case is the third axis `(axis=2)`. Here's the result of applying `np.flip(rgb, axis=2)`:

```
In [71]: np.flip(rgb, axis=2)
```

Out[71]:

```
array([[[  0,   0, 255],
        [  0,   0,   0],
        [255,   0,   0]],

       [[255,   0, 255],
        [255, 255, 255],
        [  0, 255, 255]]])
```

The elements of the array have been flipped along the third axis, which corresponds to flipping the color channels (R, G, B). So, within each pixel, the order of the color channels has been reversed. The array now represents the RGB values with the blue channel flipped to the red channel and vice versa.

Split the array into equals parts

In [83]:

```
arr1, arr2, arr3 = np.split(rgb, 3, axis=1)
```

In [84]:

```
arr1
```

Out[84]:

```
array([[[255,   0,   0]],

       [[255,   0, 255]]])
```

In [85]:

```
arr2
```

Out[85]:

```
array([[[  0,   0,   0]],

       [[255, 255, 255]]])
```

In [86]:

```
arr3
```

Out[86]:

```
array([[[  0,   0, 255]],

       [[255, 255,   0]]])
```

The code `arr1, arr2, arr3 = np.split(rgb, 3, axis=1)` splits the `rgb` array into three equal parts along the second axis `(axis=1)` and assigns them to three separate arrays: `arr1`, `arr2`, and `arr3`. Each resulting array will have a shape of `(2, 1, 3)`.

Each resulting array contains one column from the corresponding position in the original `rgb` array. Essentially, the original array has been split into three smaller arrays, each containing one column of the original data.

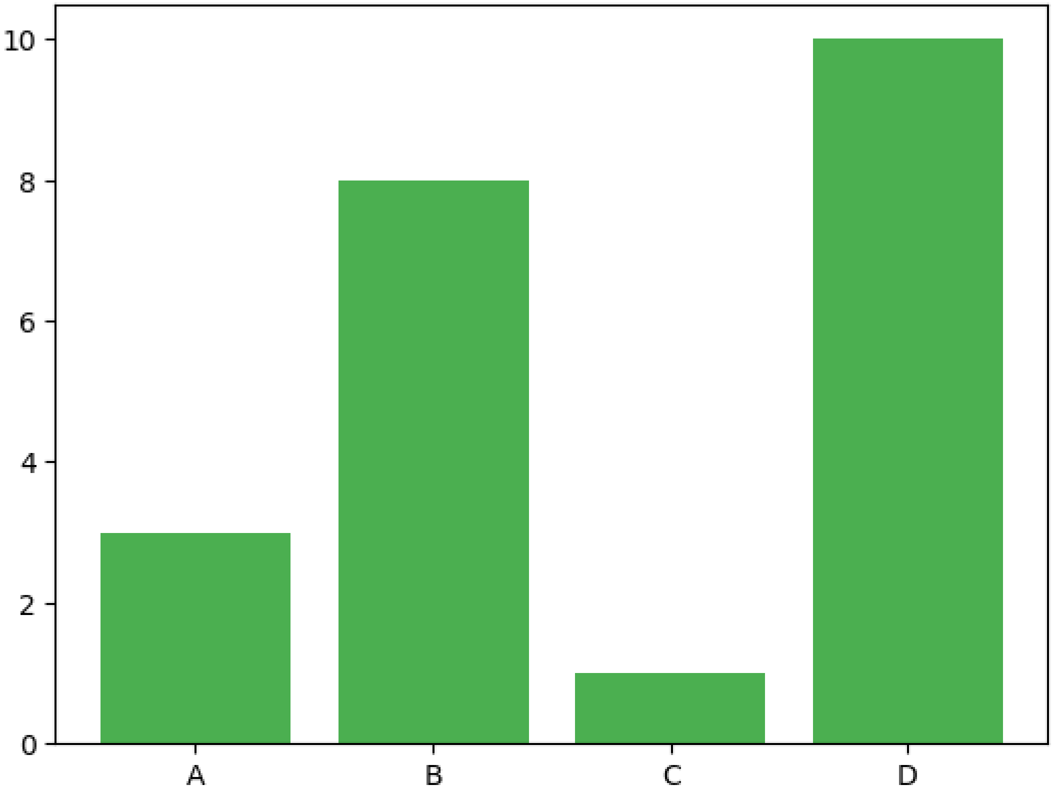
Sorting and Plotting Arrays

In [2]:

```
import matplotlib.pyplot as plt


x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
```

```
plt.bar(x, y, color="#4CAF50")
plt.show()
```

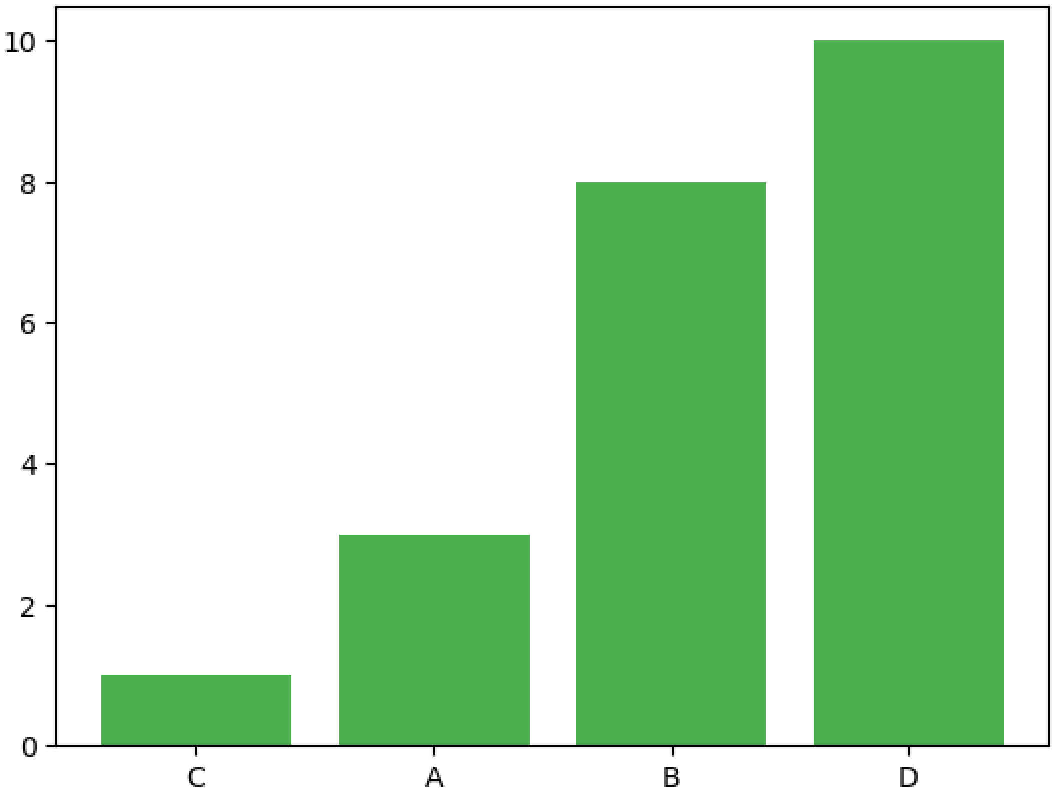


In [3]:

```
# Sorting the arrays
sorted_indices = np.argsort(y)

sorted_y = y[sorted_indices]
sorted_x = x[sorted_indices]

plt.bar(sorted_x, sorted_y, color="#4CAF50")
plt.show()
```



Arrays to Pandas Dataframe

In [8]:

```
import pandas as pd


# Create a dictonary from the x and y arrays
data = {
    "Category": x,
    "Value": y
}

# Create a dataframe from the dictionary
df = pd.DataFrame(data)

# Display the DataFrame
display(df.sort_values(by="Value", ascending=False))
```

	Category	Value
3	D	10
1	B	8
0	A	3
2	C	1

In [ ]: