S02_T02_Matrix Structure

January 28, 2022

1 IT Academy - Data Science Itinerary

1.1 S02-T02: Matrix Structure

1 :

Creates a one-dimensional np.array, including at least 8 integers, data type int64. Shows the size and shape of the array.

[221]:

```
import numpy as np

#array of 20 random integers between 0 and 500
myArray = np.random.randint(0,501,20)


print(myArray.dtype) #data type

print(myArray.size) #size of the array

print(myArray.shape) #shape of the array

print(myArray.ndim) #dimension
```

```
int64
20
(20,)
1
```

2 :

From the array in Exercise 1, calculate the mean value of the values entered and subtract the resulting average from each of the values in the array.

[222]:

```
# We can use the mean() method to calculate the mean value

mean = myArray.mean()

print(mean)

print(myArray)
```

```

# subtract the resulting average from each

print(myArray - mean)
```

```
301.65
[262 486 358 344 456  29 199 152  61 485  94 260 269 465 125 323 440 292
 441 492]
[ -39.65  184.35   56.35   42.35  154.35 -272.65 -102.65 -149.65 -240.65
  183.35 -207.65  -41.65  -32.65  163.35 -176.65   21.35  138.35   -9.65
  139.35  190.35]
```

3 :

Create a two-dimensional array with a shape of 5 x 5. Extract the maximum value of the array, and the maximum values of each of its axes.

[223]:

```
array5_5 = np.random.randint(0,101, size=(5,5))
print(array5_5)
```

```
[[100  31  36  32  75]
 [ 12  91  62  47  68]
 [ 26  55  88   9  84]
 [100  62  79  31  65]
 [  0  62  22  80  17]]
```

[224]:

```
#  the maximum value of the array
print(np.max(array5_5))



# maximum value across the rows

print(np.max(array5_5,0))

# maximum value across the columns

print(np.max(array5_5,1))
```

```
100
[100  91  88  80  84]
[100  91  88 100  80]
```

1.1.1 Level 2: Concepts of the structure of an array, broadcasting, indexing, masking.

4 :

Show examples of different arrays, the fundamental rule of Broadcasting that says, “arrays can be transmitted / broadcast if their dimensions match or if one of the arrays has a size of 1.”

[225]:

```
#Arrays M & a have equivalent first dimensions, and the second dimension of a is 1
M = np.ones((3, 3))
a = np.arange(3)

J = (M * a)

print("dimesion of J",J.shape)
print(J)
```

```
dimesion of J (3, 3)
[[0. 1. 2.]
 [0. 1. 2.]
 [0. 1. 2.]]
```

[226]:

```
##Arrays B & C have equivalent second dimensions, and the first dimension of C is 1

B = np.random.randint(0,11, size=(2,3))
C = np.random.randint(0,11, size=(1,3))

K = B * C
print("dimesion of K",K.shape)
print(K)
```

```
dimesion of K (2, 3)
[[ 6 30  0]
 [14  0  0]]
```

if arrays do not have equivalent second dimensions, and neither is 1, so operands could not be broadcast together

5 :

Use Indexing to extract the values of a column and a row from an array ans sum their values.

[227]:

```
print(array5_5)
```

```
[[100  31  36  32  75]
 [ 12  91  62  47  68]
 [ 26  55  88   9  84]
 [100  62  79  31  65]
 [  0  62  22  80  17]]
```

extracting a whole row: 1st row:

[228]:

```
row1 = array5_5[0]
print(row1)
```

```
[100  31  36  32  75]
```

extracting a whole column: 1st column:

[229]:

```
col1 = array5_5[:,0]
print(col1)
```

```
[100  12  26 100   0]
```

2nd row, 1st column:

[230]:

```
print(array5_5[1,0])
```

```
12
```

sum values from one column and one row:

[231]:

```
print(col1 + row1)
```

```
[200  43  62 132  75]
```

6 :

Mask an array by performing a vectorized Boolean calculation, taking each element and checking if it is evenly divided by four.

This returns a mask array in the same way as the element-wise results of a calculation.

[232]:

```
mask = (array5_5 % 4 == 0)

print(mask)
```

```
[[ True False  True  True False]
 [ True False False False  True]
 [False False  True False  True]
 [ True False False False False]
 [ True False False  True False]]
```

7 :

Use this mask to index the original number array. This causes the array to lose its original shape, reducing it to one dimension, but you still get the data you are looking for.

[233]:

```
print(array5_5[mask])
```

```
[100  36  32  12  68  88  84 100   0  80]
```

1.1.2 Level 3: Import of an image with Matplotlib.

8 :

Image manipulation with Matplotlib.

You will upload any image (jpg, png) with Matplotlib. note that RGB images (Red, Green, Blue) are really only widths × heights × 3 arrays (three channels Red, Green, and Blue), one for each color of int8 integers,

Manipulate these bytes and use Matplotlib again to save the modified image once you're done.

Help: Import, import matplotlib.image as mpimg. study the mpimg.imread() method

[234]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

[235]:

```
#image name = "foto.jpg"
#read it using imread()

img = mpimg.imread("foto.jpg")
```

[236]:

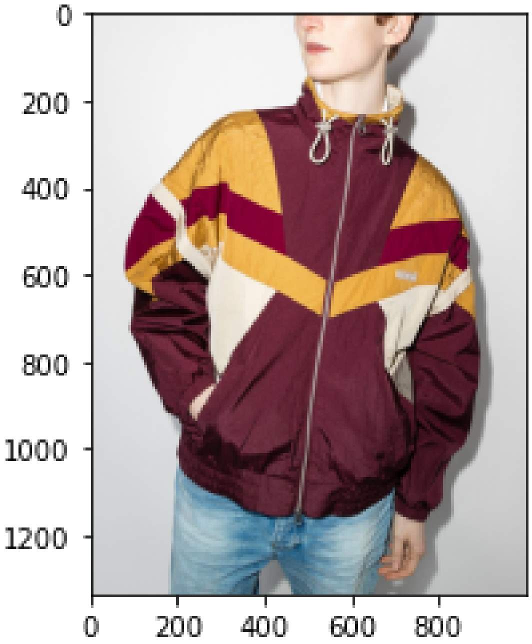
```
img.shape, img.dtype
```

[236]:

```
((1334, 1000, 3), dtype('uint8'))
```

[237]:

```
#display it using imshow ()

imgplot = plt.imshow(img)
```

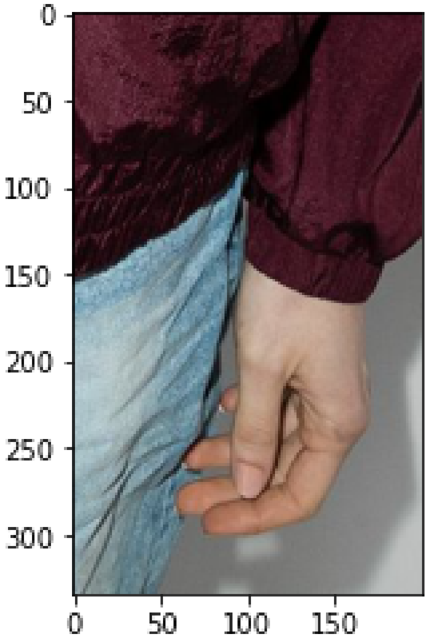


[238]:

```
#if you want to turn off the axis tricks off, call plt.axis("off")
plt.axis("off")
imgplot = plt.imshow(img)
```

```
[239]: # We can select a part of the image, using the axis as guides to slice.

       hand = img [1000:,600:800,:]
       plt.imshow(hand)
```

```
[239]: <matplotlib.image.AxesImage at 0x7f99e7215d00>
```

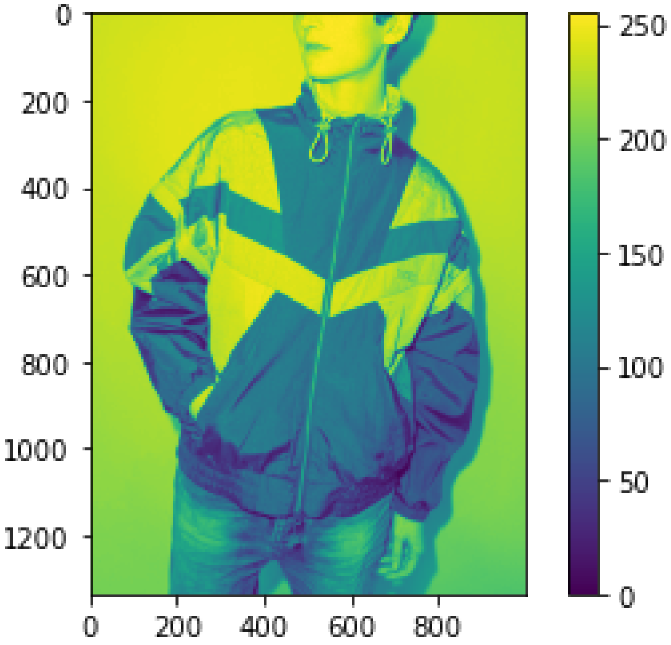


lets show RGB chanels:

- red channel:

```
[240]: red = img [:,:,0]
       plt.imshow(red)
       plt.colorbar()
```

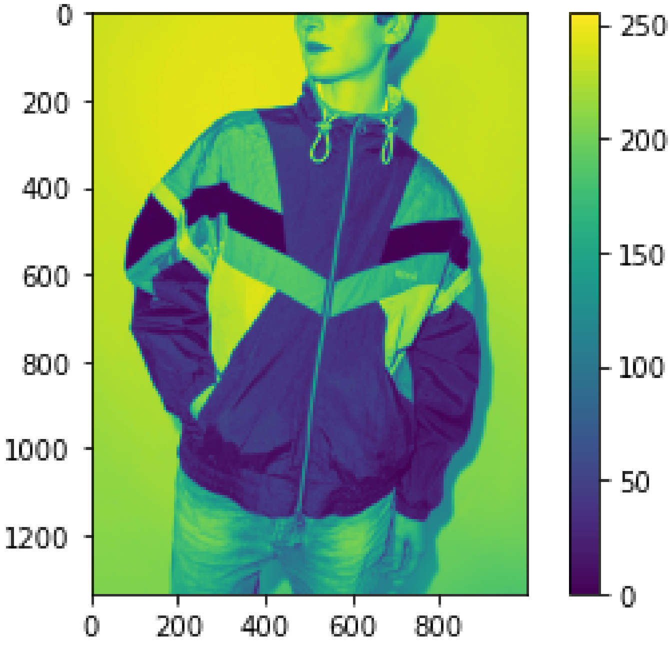
```
[240]: <matplotlib.colorbar.Colorbar at 0x7f99fcc301c0>
```



- green channel:

```
[241]: green = img [:,:,1]
       plt.imshow(green)
       plt.colorbar()
```

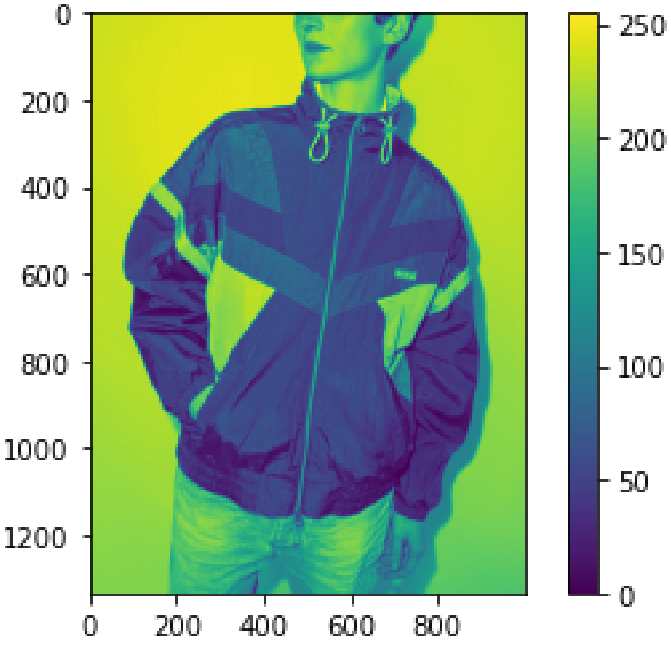
```
[241]: <matplotlib.colorbar.Colorbar at 0x7f99eed42400>
```



- blue channel:

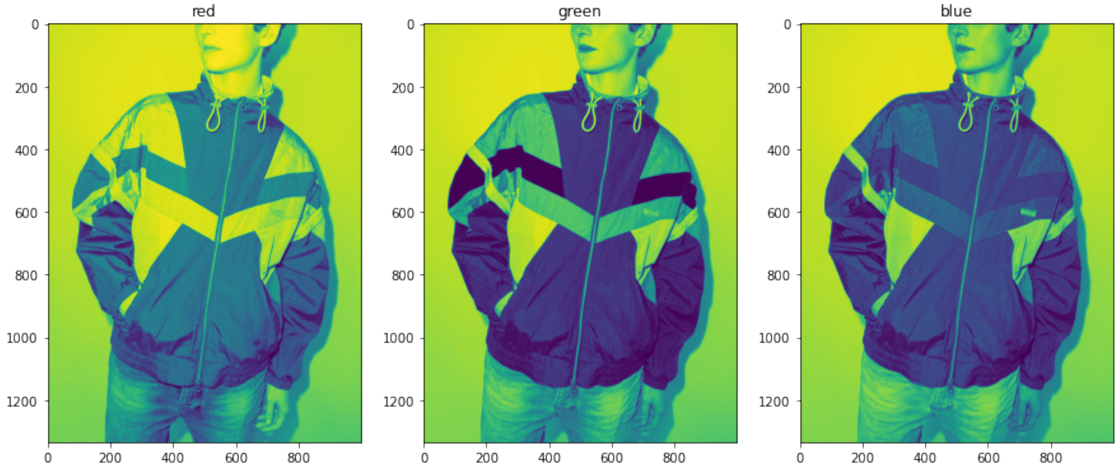
[242]:
```
blue = img [:,:,2]
plt.imshow(blue)
plt.colorbar()
```

[242]:
```
<matplotlib.colorbar.Colorbar at 0x7f99e8517670>
```



[243]:
```
fig, ax = plt.subplots(1,3,figsize=(15, 10))


ax[0].set_title('red')
ax[0].imshow(red)

ax[1].set_title('green')
ax[1].imshow(green)

ax[2].set_title('blue')
ax[2].imshow(blue)

plt.imshow
```

[243]:
```
<function matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None,
interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None,
*, filternorm=True, filterrad=4.0, resample=None, url=None, data=None,
**kwargs)>
```



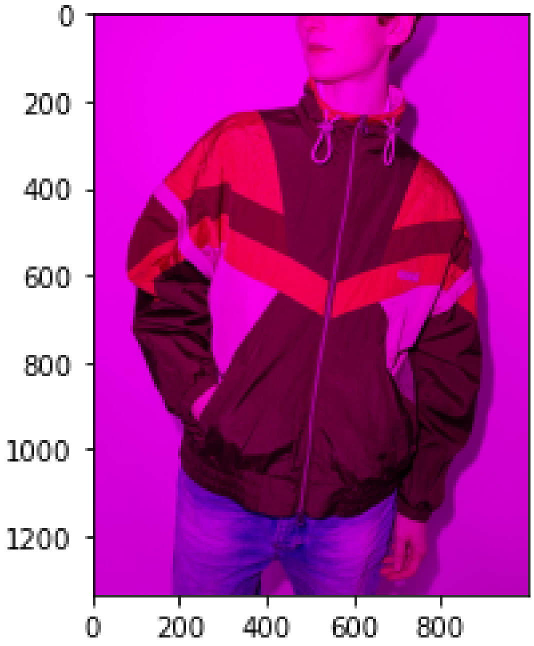
Delete green canal

[244]:
```
#copy the original image "img"
withoutgreen = img.copy()
```

[245]:
```
#Remove the green channel from the original image

withoutgreen[:,:,1] = 0
```

[246]:
```
#display modified image ( without green channel)

plt.imshow(withoutgreen)
```

[246]:
```
<matplotlib.image.AxesImage at 0x7f9a011aa730>
```



[247]:
```
# Save modified image
mpimg.imsave("withoutgreen.jpg",withoutgreen)
```

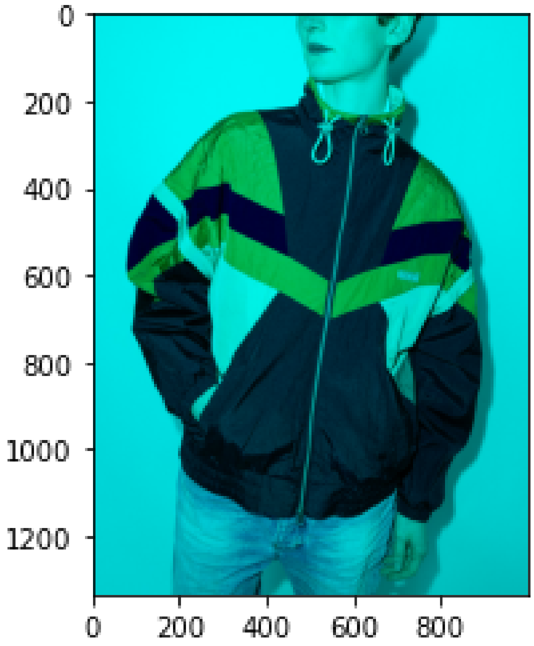
Delete red canal:

[248]:
```
withoutred = img.copy()
```

[249]:
```
#Remove the red channel from the original image

withoutred[:,:,0] = 0
```

[250]:
```
#display modified image ( without red channel)

plt.imshow(withoutred)
```

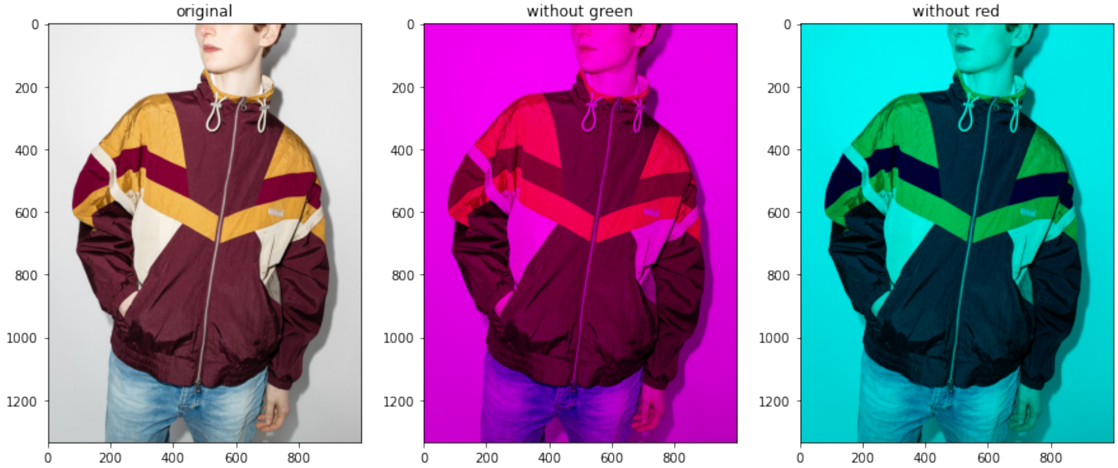
[250]:
```
<matplotlib.image.AxesImage at 0x7f99ed74cd90>
```



```
[251]: # Save modified image
       mpimg.imsave("withoutred.jpg",withoutred)
```

```
[252]: fig, ax = plt.subplots(1,3,figsize=(15, 10))
       ax[0].set_title('original')
       ax[0].imshow(img)

       ax[1].set_title('without green')
       ax[1].imshow(withoutgreen)

       ax[2].set_title('without red')
       ax[2].imshow(withoutred)
       plt.imshow
```

```
[252]: <function matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None,
       interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None,
       *, filternorm=True, filterrad=4.0, resample=None, url=None, data=None,
       **kwargs)>
```


original
without green
without red
0
200
400
600
800
1000
1200
0
200
400
600
800

[ ]: