numpy-lecture4-dec-batch

May 30, 2023

0.1 Numpy Lecture - 4

[2]:
```
import numpy as np
```

[3]:
```
a = np.array([2, 30, 41, 7, 19, 25])
a.ndim
```

[3]:
```
1
```

[4]:
```
np.sort(a)
```

[4]:
```
array([ 2,  7, 19, 25, 30, 41])
```

[5]:
```
a
```

[5]:
```
array([ 2, 30, 41,  7, 19, 25])
```

[6]:
```
a.sort()
```

[7]:
```
a
```

[7]:
```
array([ 2,  7, 19, 25, 30, 41])
```

[8]:
```
a = np.arange(9, 0, -1).reshape(3, 3)
a
```

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

[9]:
```
np.sort(a, axis=0)
```

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

[10]:
```
np.sort(a)
```

[10]:
```
array([[7, 8, 9],
       [4, 5, 6],
       [1, 2, 3]])
```

[11]:
```
help(np.sort)
```

```
Help on function sort in module numpy:

sort(a, axis=-1, kind=None, order=None)
    Return a sorted copy of an array.

    Parameters
    ----------
    a : array_like
        Array to be sorted.
    axis : int or None, optional
        Axis along which to sort. If None, the array is flattened before
        sorting. The default is -1, which sorts along the last axis.
    kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional
        Sorting algorithm. The default is 'quicksort'. Note that both 'stable'
        and 'mergesort' use timsort or radix sort under the covers and, in
general,
        the actual implementation will vary with data type. The 'mergesort'
option
        is retained for backwards compatibility.

        .. versionchanged:: 1.15.0.
           The 'stable' option was added.

    order : str or list of str, optional
        When `a` is an array with fields defined, this argument specifies
        which fields to compare first, second, etc.  A single field can
        be specified as a string, and not all fields need be specified,
        but unspecified fields will still be used, in the order in which
        they come up in the dtype, to break ties.

    Returns
    -------
    sorted_array : ndarray
        Array of the same type and shape as `a`.

    See Also
    --------
    ndarray.sort : Method to sort an array in-place.
    argsort : Indirect sort.
    lexsort : Indirect stable sort on multiple keys.
    searchsorted : Find elements in a sorted array.
    partition : Partial sort.

    Notes
    -----
    The various sorting algorithms are characterized by their average speed,
    worst case performance, work space size, and whether they are stable. A
    stable sort keeps items with the same key in the same relative
    order. The four algorithms implemented in NumPy have the following
    properties:

    =========== ======= ============= ============ ========
       kind      speed   worst case    work space   stable
    =========== ======= ============= ============ ========
    'quicksort'    1     O(n^2)            0          no
    'heapsort'     3     O(n*log(n))       0          no
    'mergesort'    2     O(n*log(n))      ~n/2        yes
    'timsort'      2     O(n*log(n))      ~n/2        yes
    =========== ======= ============= ============ ========

    .. note:: The datatype determines which of 'mergesort' or 'timsort'
       is actually used, even if 'mergesort' is specified. User selection
       at a finer scale is not currently available.

    All the sort algorithms make temporary copies of the data when
    sorting along any but the last axis.  Consequently, sorting along
    the last axis is faster and uses less space than sorting along
    any other axis.

    The sort order for complex numbers is lexicographic. If both the real
    and imaginary parts are non-nan then the order is determined by the
    real parts except when they are equal, in which case the order is
    determined by the imaginary parts.

    Previous to numpy 1.4.0 sorting real and complex arrays containing nan
    values led to undefined behaviour. In numpy versions >= 1.4.0 nan
    values are sorted to the end. The extended sort order is:

      * Real: [R, nan]
      * Complex: [R + Rj, R + nanj, nan + Rj, nan + nanj]

    where R is a non-nan real value. Complex values with the same nan
    placements are sorted according to the non-nan part if it exists.
    Non-nan values are sorted as before.

    .. versionadded:: 1.12.0

    quicksort has been changed to `introsort
<https://en.wikipedia.org/wiki/Introsort>`_.
```

```
    When sorting does not make enough progress it switches to
    `heapsort <https://en.wikipedia.org/wiki/Heapsort>`_.
    This implementation makes quicksort O(n*log(n)) in the worst case.

    'stable' automatically chooses the best stable sorting algorithm
    for the data type being sorted.
    It, along with 'mergesort' is currently mapped to
    `timsort <https://en.wikipedia.org/wiki/Timsort>`_
    or `radix sort <https://en.wikipedia.org/wiki/Radix_sort>`_
    depending on the data type.
    API forward compatibility currently limits the
    ability to select the implementation and it is hardwired for the different
    data types.

    .. versionadded:: 1.17.0

    Timsort is added for better performance on already or nearly
    sorted data. On random data timsort is almost identical to
    mergesort. It is now used for stable sort while quicksort is still the
    default sort if none is chosen. For timsort details, refer to
    `CPython listsort.txt
<https://github.com/python/cpython/blob/3.7/Objects/listsort.txt>`_.
    'mergesort' and 'stable' are mapped to radix sort for integer data types.
Radix sort is an
    O(n) sort instead of O(n log n).

    .. versionchanged:: 1.18.0

    NaT now sorts to the end of arrays for consistency with NaN.

    Examples
    --------
    >>> a = np.array([[1,4],[3,1]])
    >>> np.sort(a)                # sort along the last axis
    array([[1, 4],
           [1, 3]])
    >>> np.sort(a, axis=None)     # sort the flattened array
    array([1, 1, 3, 4])
    >>> np.sort(a, axis=0)        # sort along the first axis
    array([[1, 1],
           [3, 4]])

    Use the `order` keyword to specify a field to use when sorting a
    structured array:

    >>> dtype = [('name', 'S10'), ('height', float), ('age', int)]
    >>> values = [('Arthur', 1.8, 41), ('Lancelot', 1.9, 38),
    …           ('Galahad', 1.7, 38)]
```

```
>>> a = np.array(values, dtype=dtype)       # create a structured array
>>> np.sort(a, order='height')                        # doctest: +SKIP
array([('Galahad', 1.7, 38), ('Arthur', 1.8, 41),
       ('Lancelot', 1.8999999999999999, 38)],
      dtype=[('name', '|S10'), ('height', '<f8'), ('age', '<i4')])

Sort by age, then height if ages are equal:

>>> np.sort(a, order=['age', 'height'])               # doctest: +SKIP
array([('Galahad', 1.7, 38), ('Lancelot', 1.8999999999999999, 38),
       ('Arthur', 1.8, 41)],
      dtype=[('name', '|S10'), ('height', '<f8'), ('age', '<i4')])
```

```
[12]: a = np.array([2, 30, 41, 7, 19, 25])
      np.argsort(a)
```

```
[12]: array([0, 3, 4, 5, 1, 2])
```

0.1.1 Matrix Multiplication

```
[13]: a = np.arange(5)
      b = np.ones(5)*2
```

```
[14]: a*b
```

```
[14]: array([0., 2., 4., 6., 8.])
```

```
[15]: a = np.arange(12).reshape(3, 4)
      b = np.arange(12).reshape(3, 4)
```

```
[16]: a*b
```

```
[16]: array([[  0,   1,   4,   9],
             [ 16,  25,  36,  49],
             [ 64,  81, 100, 121]])
```

```
[17]: b = b.T
```

```
[18]: np.matmul(a, b)
```

```
[18]: array([[ 14,  38,  62],
             [ 38, 126, 214],
             [ 62, 214, 366]])
```

```
[19]: a @ b
```

```
[19]: array([[ 14,  38,  62],
             [ 38, 126, 214],
             [ 62, 214, 366]])
```

```
[20]: np.dot(a, b)
```

```
[20]: array([[ 14,  38,  62],
             [ 38, 126, 214],
             [ 62, 214, 366]])
```

```
[21]: c = np.array([1, 2, 3])
      d = np.array([5, 6, 7])
      np.dot(c, d)
```

```
[21]: 38
```

```
[22]: np.dot(4, 5)
```

```
[22]: 20
```

```
[23]: a = np.arange(12).reshape(3, 4)
      b = np.arange(12).reshape(3, 4)
```

```
[24]: a @ b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [24], in <cell line: 1>()
----> 1 a @ b

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, wit
  ↪gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 4)
```

```
[25]: a = np.arange(10)
      a*2
```

```
[25]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
[26]: a = np.array([[1, 2, 3], [4, 5, 6]])
      a*2
```

```
[26]: array([[ 2,  4,  6],
             [ 8, 10, 12]])
```

```
[27]: import math
```

[28]:
```
a = np.arange(1, 11)
math.log(a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [28], in <cell line: 2>()
      1 a = np.arange(1, 11)
----> 2 math.log(a)

TypeError: only size-1 arrays can be converted to Python scalars
```

[29]:
```
np_vect = np.vectorize(math.log)
np_vect
```

[29]:
```
<numpy.vectorize at 0x113ed5580>
```

[30]:
```
np_vect(a)
```

[30]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

[31]:
```
np_vect = np.vectorize(math.log)(a)
np_vect
```

[31]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

0.1.2 3D Array

[32]:
```
a = np.arange(24).reshape(2, 3, 4)
a
```

[32]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[33]:
```
a[0, 1, 1]
```

[33]:
```
5
```

[34]:
```
a[1, 1, 2]
```

[34]: 18

0.1.3 Image Manipulation

[35]:
```
!gdown 17tYTDPBU5hpby9t0kGd7w_-zBsbY7sEd
```

```
Downloading…
From: https://drive.google.com/uc?id=17tYTDPBU5hpby9t0kGd7w_-zBsbY7sEd
To: /Users/satish/Desktop/scaler/Dec Tue Batch - DAV-1/fruits.png
100%|                    | 4.71M/4.71M [00:00<00:00, 9.09MB/s]
```

[36]:
```
!gdown 1o-8yqdTM7cfz_mAaNCi2nH0urFu7pcqI
```

```
Downloading…
From: https://drive.google.com/uc?id=1o-8yqdTM7cfz_mAaNCi2nH0urFu7pcqI
To: /Users/satish/Desktop/scaler/Dec Tue Batch - DAV-1/emma_stone.jpeg
100%|                    | 80.3k/80.3k [00:00<00:00, 3.31MB/s]
```

[37]:
```
!pip install matplotlib
```

```
DEPRECATION: Configuring installation scheme with distutils config files is

deprecated and will no longer work in the near future. If you are using a

Homebrew or Linuxbrew Python, please see discussion at

https://github.com/Homebrew/homebrew-core/issues/76621

Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/site-
packages (3.5.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.9/site-packages (from matplotlib) (4.30.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.9/site-packages (from matplotlib) (1.4.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/site-
packages (from matplotlib) (1.22.3)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/site-
packages (from matplotlib) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/site-
packages (from matplotlib) (9.0.1)
Requirement already satisfied: pyparsing>=2.2.1 in
/usr/local/lib/python3.9/site-packages (from matplotlib) (3.0.7)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.9/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
DEPRECATION: Configuring installation scheme with distutils config files is
deprecated and will no longer work in the near future. If you are using a
Homebrew or Linuxbrew Python, please see discussion at
https://github.com/Homebrew/homebrew-core/issues/76621
```

[38]:

```
import matplotlib.pyplot as plt
```

[39]:

```
img = plt.imread('fruits.png')
```

[40]:

```
type(img)
```

[40]:

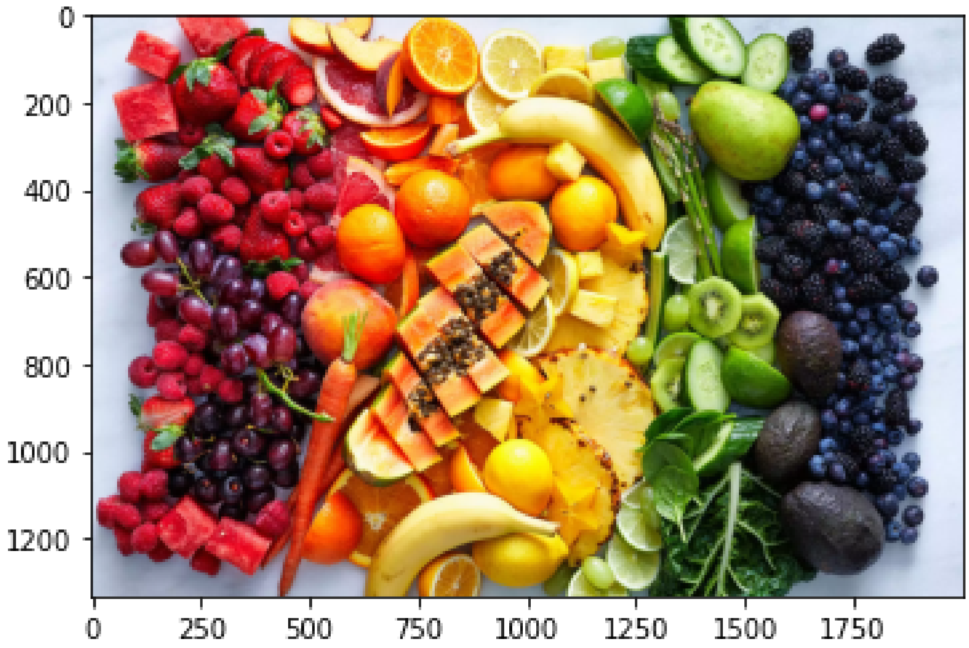
```
numpy.ndarray
```

[41]:

```
plt.imshow(img)
```

[41]:

```
<matplotlib.image.AxesImage at 0x11f356310>
```



[42]:

```
img
```

[42]:

```
array([[[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        …,
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ]],

       [[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        …,
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ]],

       [[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        …,
        [0.8039216 , 0.85882354, 0.9137255 ],
        [0.8039216 , 0.85882354, 0.9137255 ],
        [0.8039216 , 0.85882354, 0.9137255 ]],

       …,

       [[0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        …,
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ]],

       [[0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        …,
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ]],

       [[0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        [0.74509805, 0.79607844, 0.87058824],
        …,
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ],
        [0.83137256, 0.8627451 , 0.9411765 ]]], dtype=float32)
```

```
[43]: img.ndim
```

```
[43]: 3
```

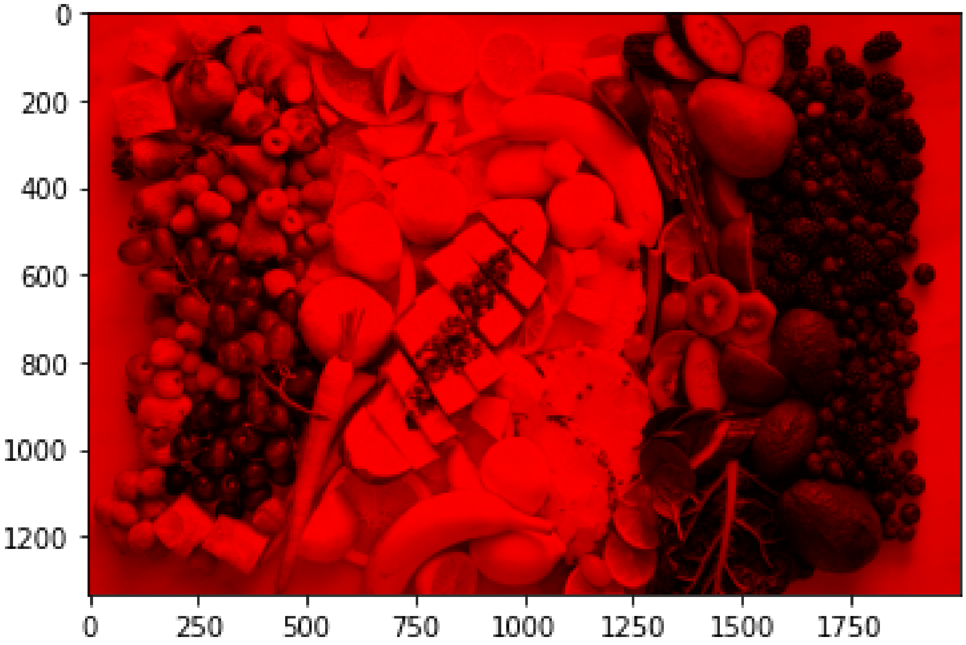
```
[44]: img.shape
```

```
[44]: (1333, 2000, 3)
```

```
[45]: img_r = img.copy()
```

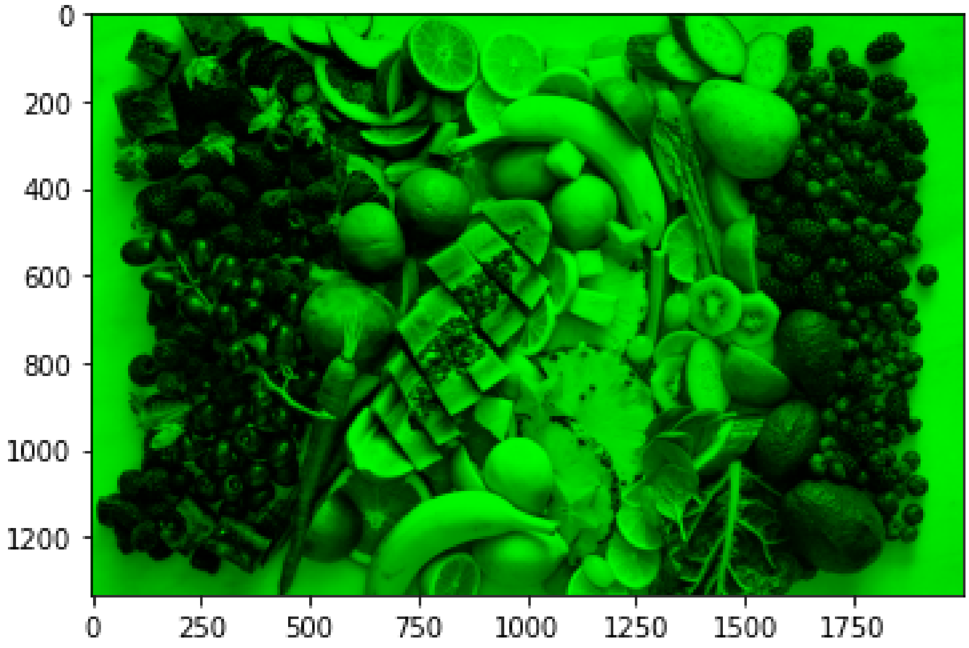
```
[46]: img_r[:, :, (1, 2)] = 0
      plt.imshow(img_r)
```

```
[46]: <matplotlib.image.AxesImage at 0x11f49e5e0>
```



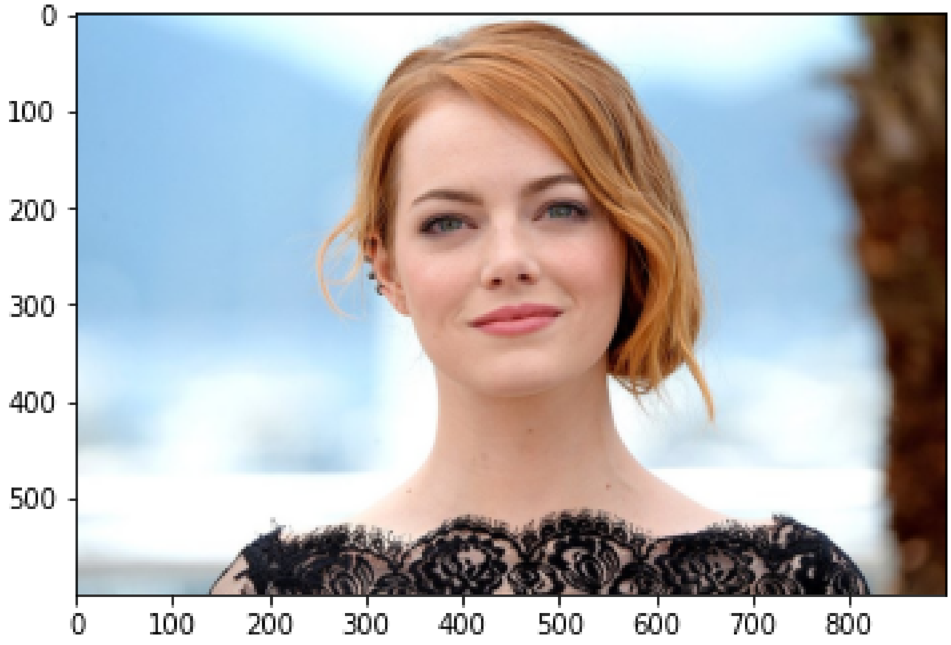
```
[48]: img_g = img.copy()
      img_g[:, :, (0, 2)] = 0
      plt.imshow(img_g)
```

```
[48]: <matplotlib.image.AxesImage at 0x11f54c910>
```



[49]:
```
img_emma = plt.imread('emma_stone.jpeg')
plt.imshow(img_emma)
```

[49]:
```
<matplotlib.image.AxesImage at 0x11f602e50>
```



[50]:
```
img_emma.shape
```

[50]:
```
(600, 900, 3)
```

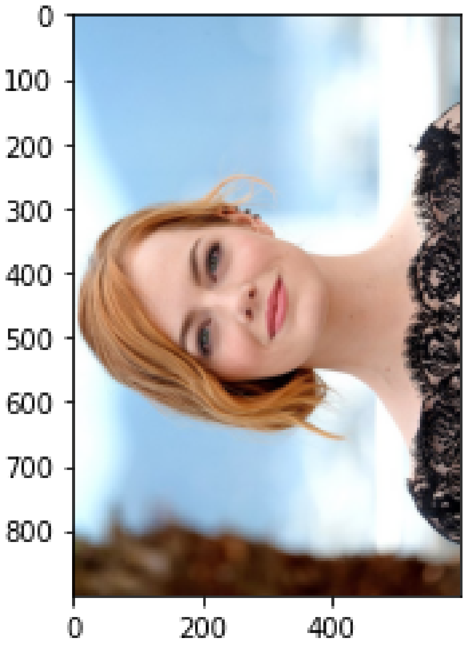
[51]:
```
help(np.transpose)
```

```
Help on function transpose in module numpy:

transpose(a, axes=None)
    Reverse or permute the axes of an array; returns the modified array.

    For an array a with two axes, transpose(a) gives the matrix transpose.

    Refer to `numpy.ndarray.transpose` for full documentation.

    Parameters
    ----------
    a : array_like
        Input array.
    axes : tuple or list of ints, optional
        If specified, it must be a tuple or list which contains a permutation of
        [0,1,..,N-1] where N is the number of axes of a.  The i'th axis of the
        returned array will correspond to the axis numbered ``axes[i]`` of the
        input.  If not specified, defaults to ``range(a.ndim)[::-1]``, which
        reverses the order of the axes.

    Returns
    -------
    p : ndarray
        `a` with its axes permuted.  A view is returned whenever
        possible.

    See Also
    --------
    ndarray.transpose : Equivalent method
    moveaxis
    argsort

    Notes
    -----
    Use `transpose(a, argsort(axes))` to invert the transposition of tensors
    when using the `axes` keyword argument.

    Transposing a 1-D array returns an unchanged view of the original array.

    Examples
    --------
```

```
>>> x = np.arange(4).reshape((2,2))
>>> x
array([[0, 1],
       [2, 3]])

>>> np.transpose(x)
array([[0, 2],
       [1, 3]])

>>> x = np.ones((1, 2, 3))
>>> np.transpose(x, (1, 0, 2)).shape
(2, 1, 3)

>>> x = np.ones((2, 3, 4, 5))
>>> np.transpose(x).shape
(5, 4, 3, 2)
```

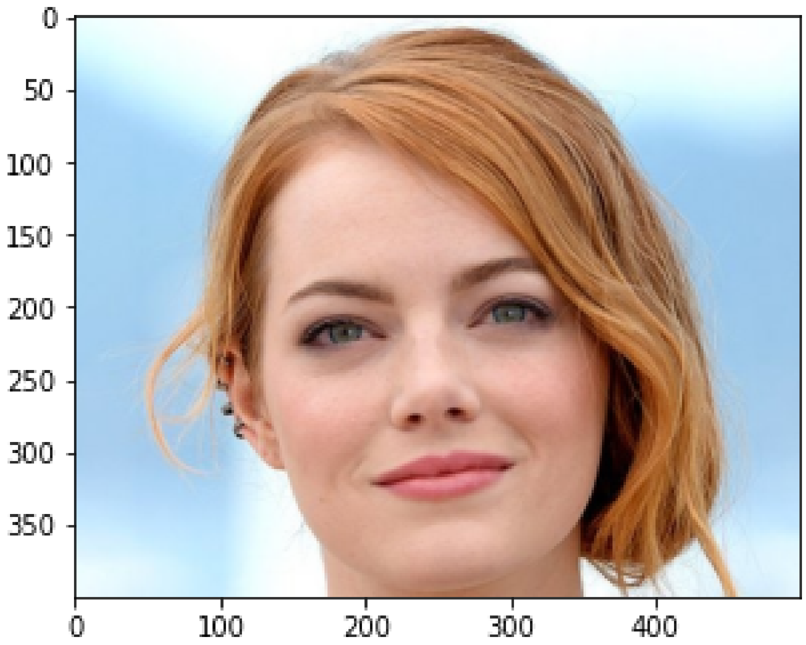
[52]:
```
img_rotated = np.transpose(img_emma, (1, 0, 2))
plt.imshow(img_rotated)
```

[52]:
```
<matplotlib.image.AxesImage at 0x11f4dbfd0>
```



[53]:
```
img_crop = img_emma[:400, 200:700, :]
plt.imshow(img_crop)
```

```
[53]: <matplotlib.image.AxesImage at 0x11f645a00>
```



```
[54]: plt.imsave('emma_trim.jpg', img_crop)
```

```
[55]: a = np.arange(4)
      b = a.reshape(2, 2)
      a[0] = 100
      b
```

```
[55]: array([[100,   1],
             [  2,   3]])
```

```
[56]: a
```

```
[56]: array([100,   1,   2,   3])
```

```
[ ]:
```