Image Processing in Python

Matthew Lowe

Matthew.Lowe2@christie.nhs.uk

MANCHESTER 1824

The Christie NHS
NHS Foundation Trust

In this session we will learn about:

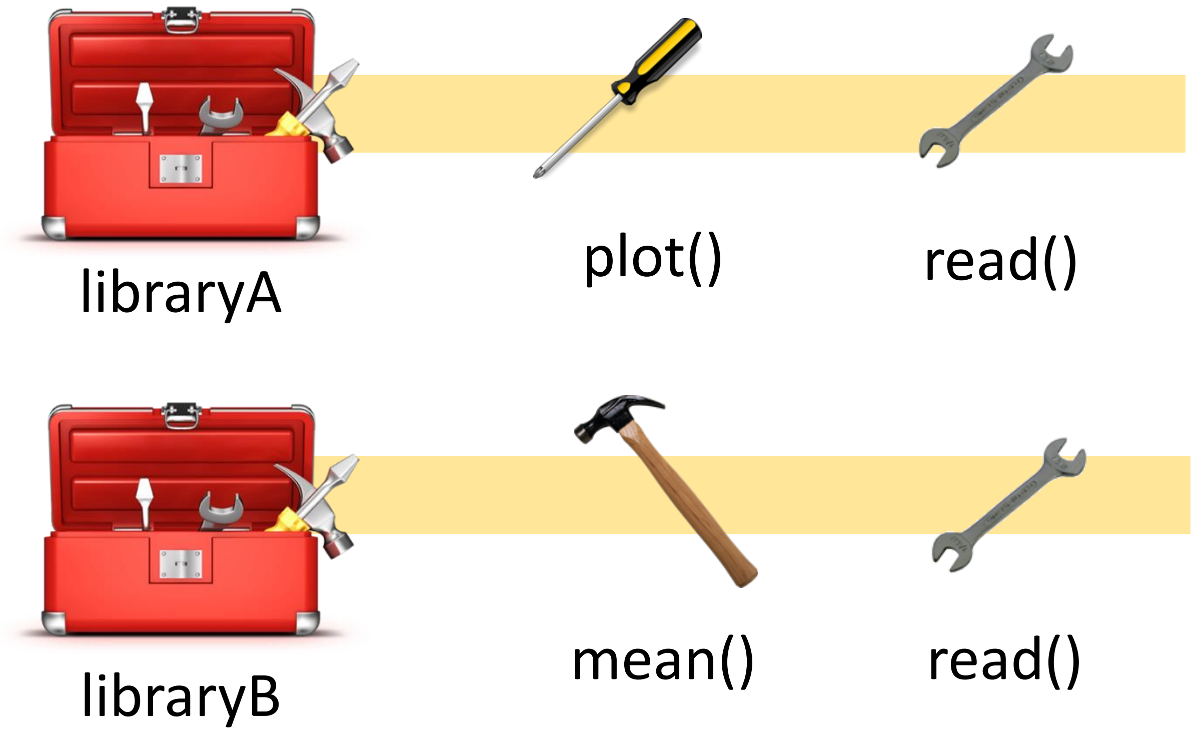
- Loading images
- Using colourmaps
- Zooming/cropping
- Shifting images
- Applying masks
- Flipping images
- Rotating images
- Blurring images
- Overlaying images
- Using transparency
- Resizing images
- Setting different window levels

You will be able to try these out during this session using the workbook.

Feel free to experiment to help you understand what is being done.

First let's import some useful libraries

```
import matplotlib.pyplot as plt
import numpy as np

from skimage import io
```



Function: imread belongs to the ‘io’ namespace

```python
image = skimage.io.imread("bird.jpg")
```

Variable name:
Could be anything. Try to pick something meaningful. Ideally, function names should be lowercase, with words separated by underscores as necessary to improve readability.

Function arguments:
In this case it is the file that we want to open.
Because it is in the same folder we can just write the name of the file. Otherwise we would write the path to the image.

Function: imshow belongs to the ‘plt’ namespace. It displays an image on some axes

```
plt.imshow(image)
```

Function arguments:
In this case it is the variable holding the image data.
Lots of other arguments are available. We’ll see some later.
http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.imshow

```
plt.show()
```

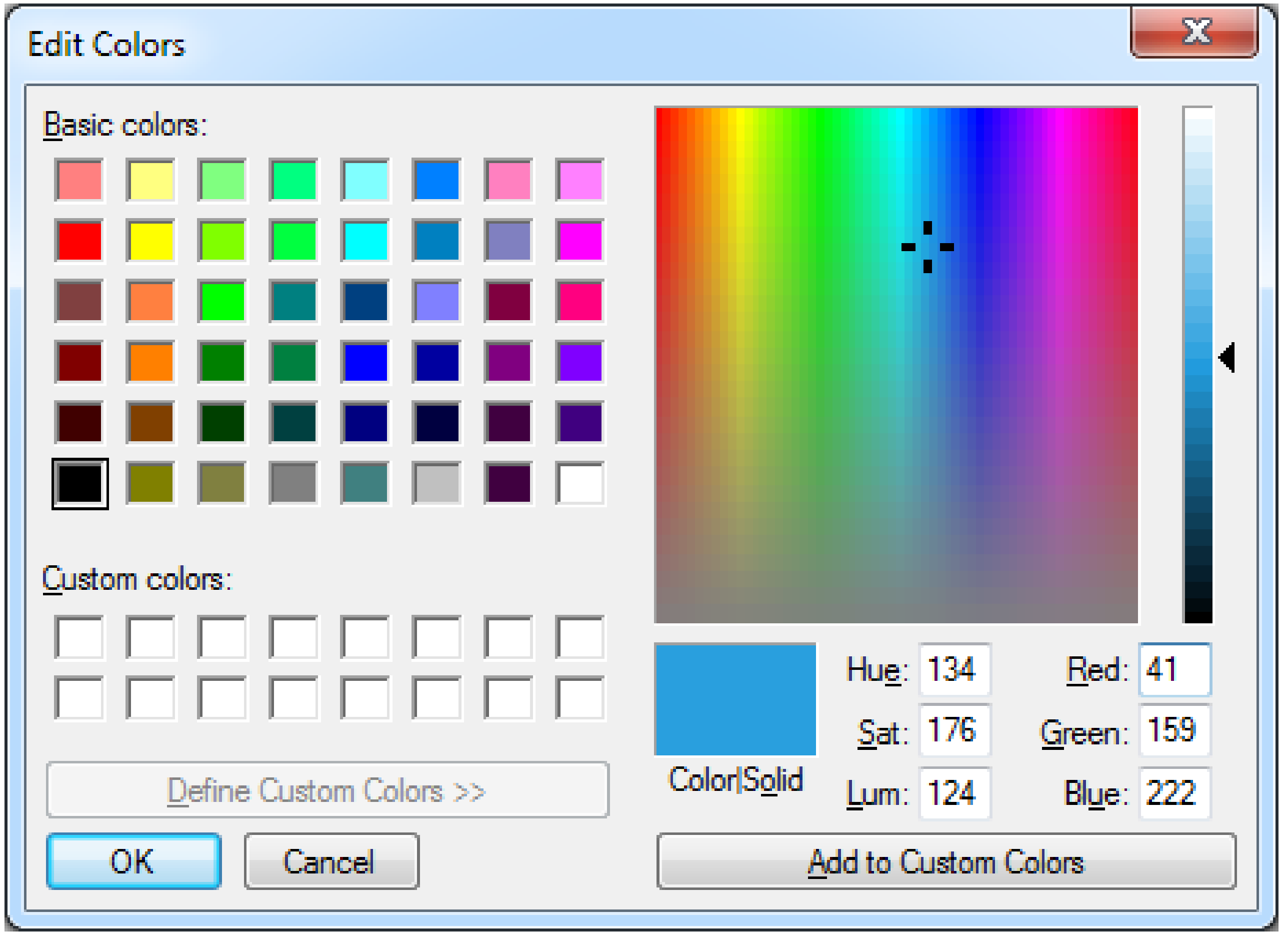
Displays the figure

```
print image
print image.shape
```

$$\underbrace{\texttt{image.shape}}_{\text{Array dimensions}}$$

An image is just an array of numbers.

What is the shape of the array?
Why does it have this shape?



Edit Colors

Hue: 134
Sat: 176
Lum: 124
Red: 41
Green: 159
Blue: 222

New variable

We want to copy the array that is stored in the variable "image"

```
blue_image = image.copy()

blue_image[:, :, 0] = 0
```

blue_image[row, column, channel] = 0

This sets the first channel in every pixel to be zero

Remember:

[1:5] is equivalent to "from 1 to 5" (5 not included)

[1:] is equivalent to "1 to the end"

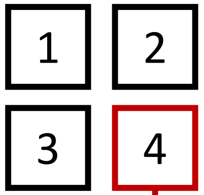
[:5] is equivalent to "from the start to 5" (5 not included)

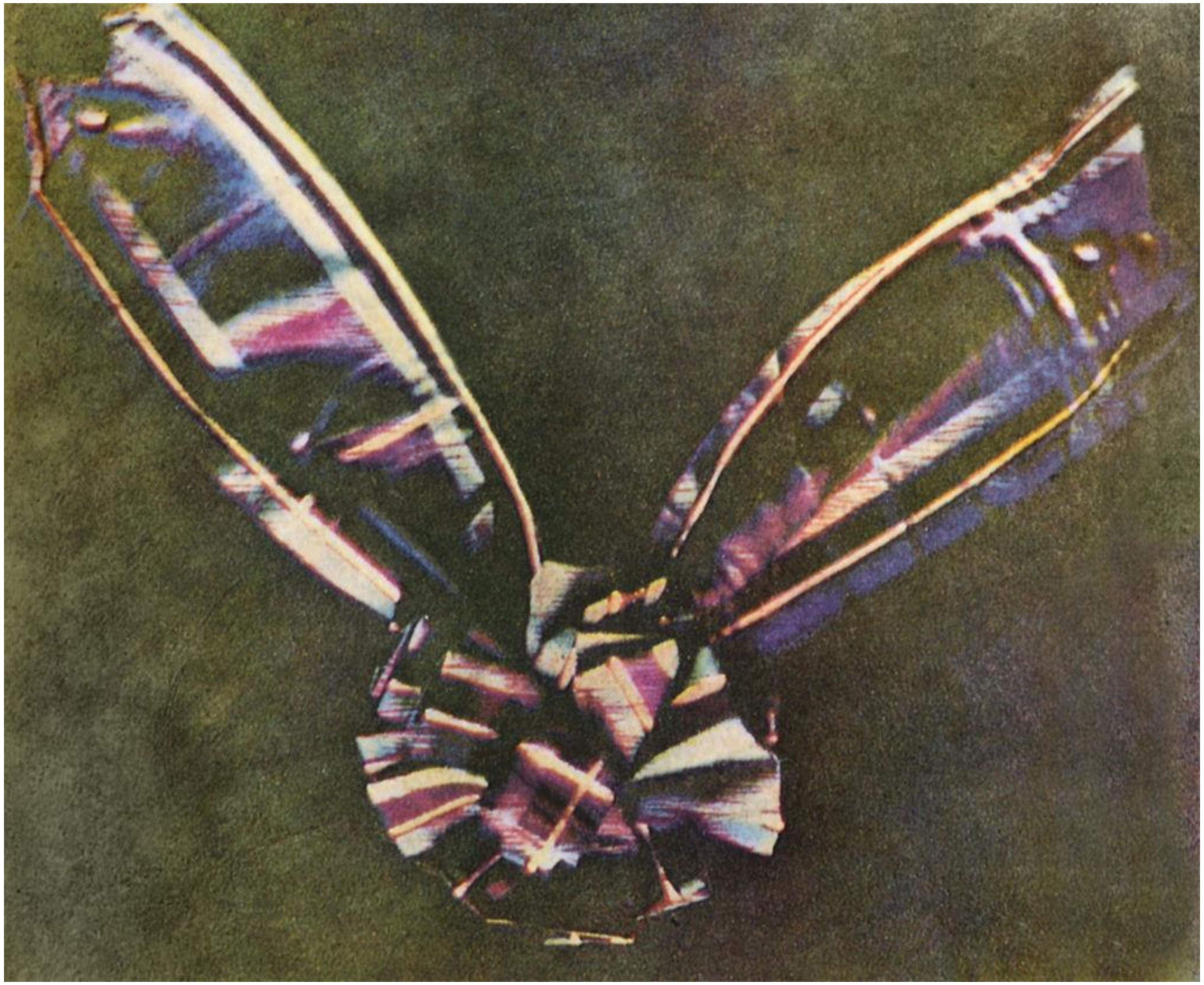
So, [:] means from the start to the end

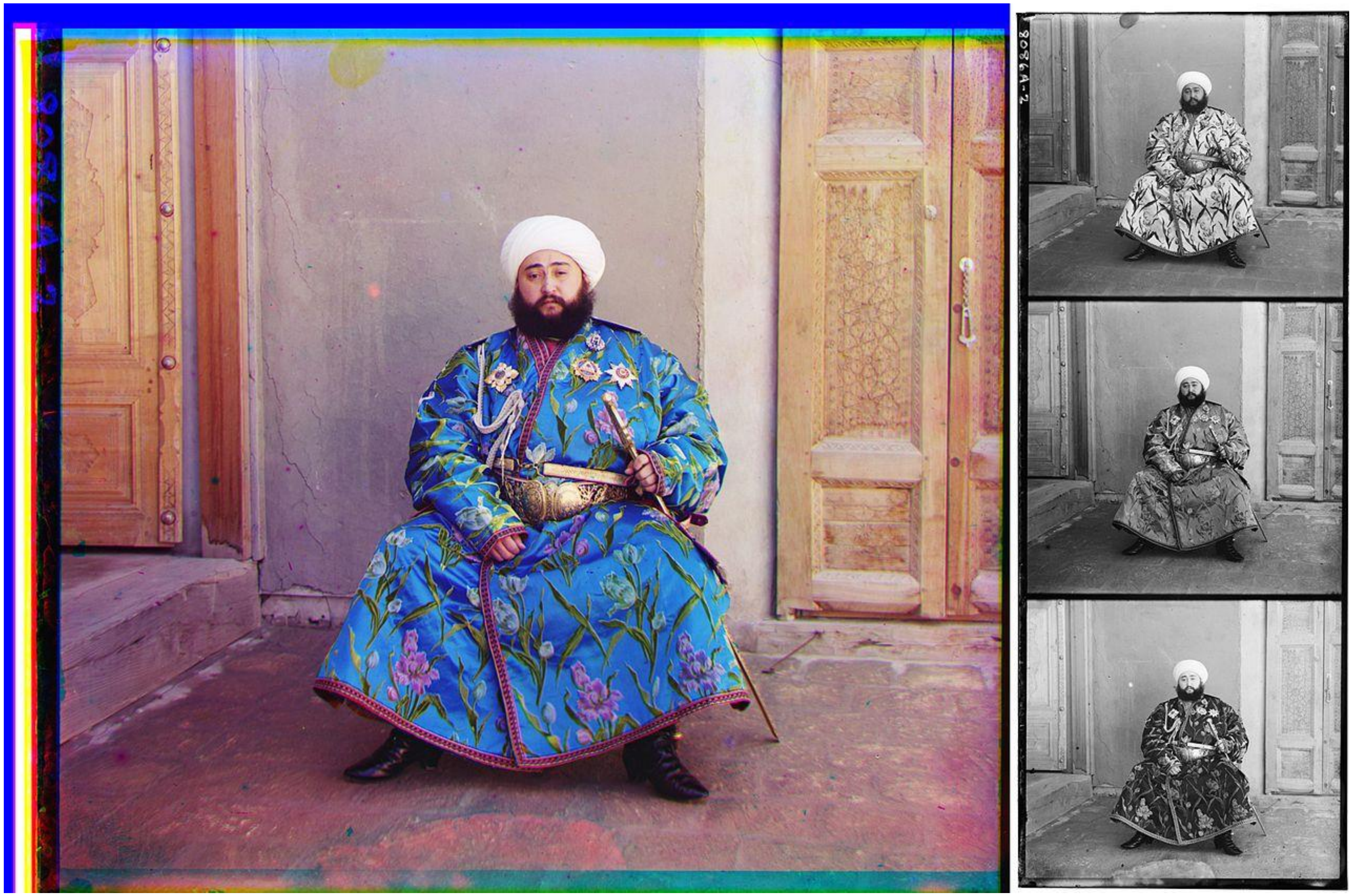
Creates a new figure as a variable called "fig"

```
fig = plt.figure()

ax = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)

ax.imshow(image)
ax2.imshow(red_image)
ax3.imshow(blue_image)
ax4.imshow(green_image)


plt.show()
```

Add some axes
Here, there are 2x2 subplots







A photograph of Mohammed Alim Khan (1880–1944), Emir of Bukhara, taken in 1911 by Sergei Mikhailovich Prokudin-Gorskii using three exposures with blue, green, and red filters.

We have 3 colour channels (In some cases we might have 4 channels, in this case the last one is transparency)
Let’s simplify things and just have one value for each pixel. This value should contain the mean value over the three channels.
We will use the “mean” function from numpy (Remember we imported it as “np” earlier)
With numpy we can do functions on arrays very efficiently without having to loop through every element

```
mean_image = np.mean(image, -1)
```

Function arguments: The first argument here is the array we want to calculate the mean over. The second argument is an axis we are calculating this over. -1 means the last one, we could put a 2 here instead.

```
a = np.linspace(0,24,25)
a.shape=(5,5)
```

What is the output of:

```
np.mean(a)
np.mean(a,0)
np.mean(a,1)
np.mean(a,-1)
```

We’ve now averaged over all three colour channels*.

What happens if we display this array now?

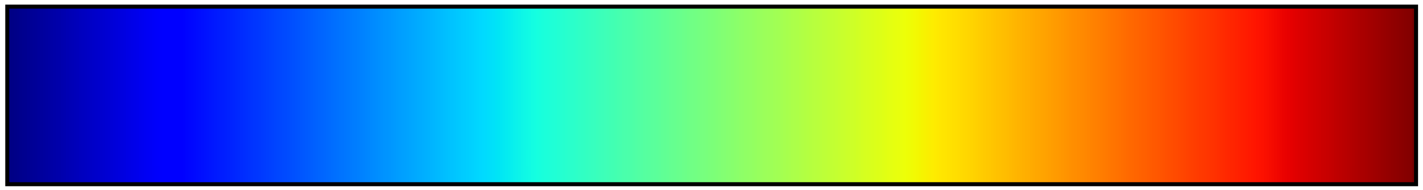
Why does it look like this?

*There’s an easier way to flatten the image though:

```
image = io.imread("bird.jpg‶, as_gray=True)
```

The image is now using the default colourmap.

A colour map translates numbers into different colours. The default colour map (called “jet”) looks like this:



Low value High value

Is this a good colourmap?

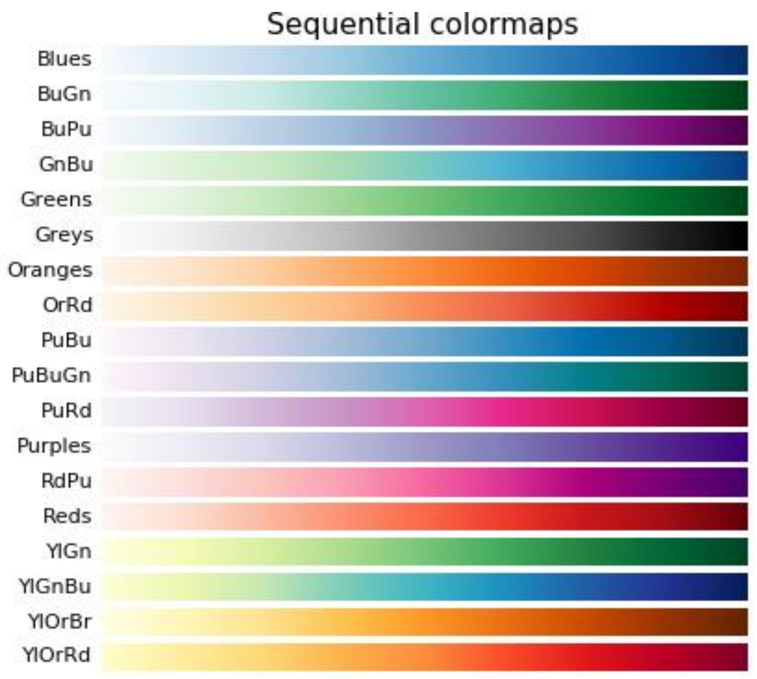
Let’s explore some other colourmaps

Try some of these colour maps:

viridis, inferno, plasma, magma, Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd, afmhot, autumn, bone, cool, copper, gist_heat, gray, hot, pink, spring, summer, winter, BrBG, bwr, coolwarm, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral, seismic, Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3, gist_earth, terrain, ocean, gist_stern, brg, CMRmap, cubehelix, gnuplot, gnuplot2, gist_ncar, nipy_spectral, jet, rainbow, gist_rainbow, hsv, flag, prism

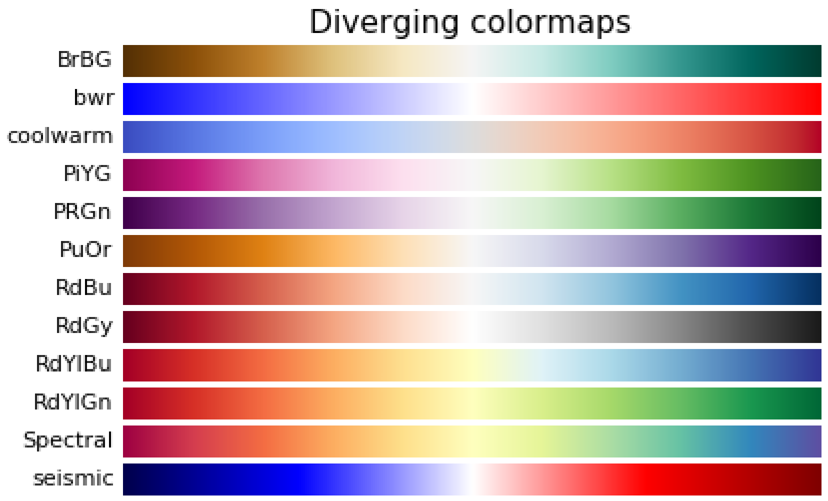
What happens if you put "_r" after the name (e.g. "Greys_r")?

Sequential: These colourmaps are approximately monochromatic colourmaps varying smoothly between two colour tones-usually from low saturation (e.g. white) to high saturation (e.g. a bright blue). Sequential colourmaps are ideal for representing most scientific data since they show a clear progression from low-to-high values.



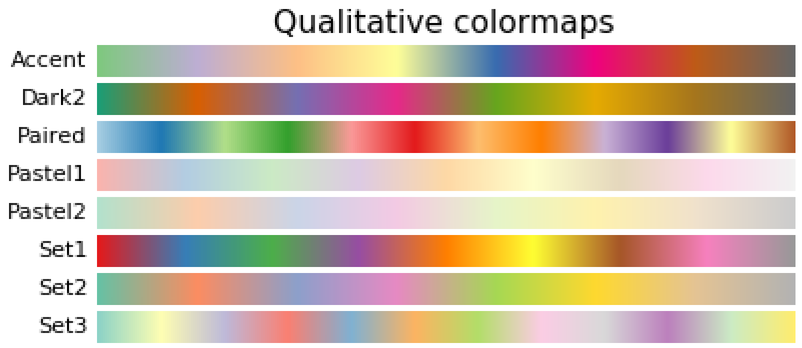
http://matplotlib.org/users/colormaps.html

Diverging: These colourmaps have a median value (usually light in colour) and vary smoothly to two different colour tones at high and low values. Diverging colourmaps are ideal when your data has a median value that is significant (e.g. 0, such that positive and negative values are represented by different colours of the colourmap).



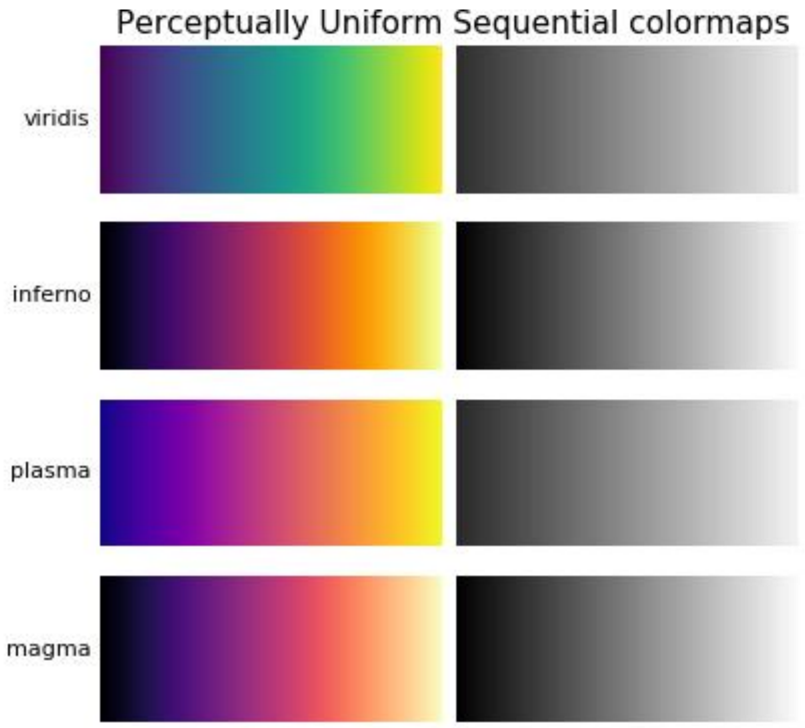
http://matplotlib.org/users/colormaps.html

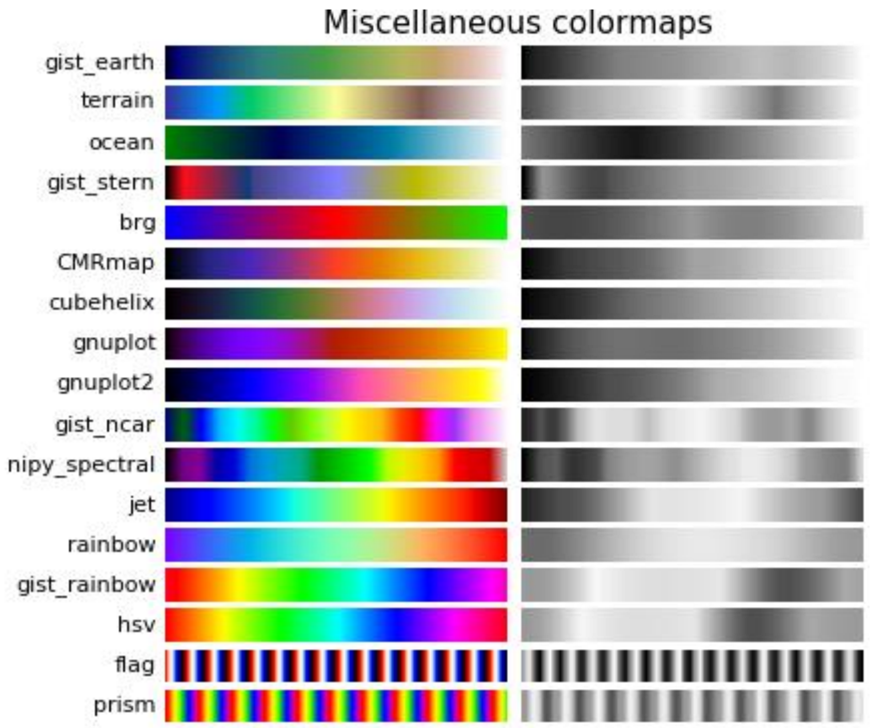
Qualitative: often are miscellaneous colours; should be used to represent information which does not have ordering or relationships.



http://matplotlib.org/users/colormaps.html

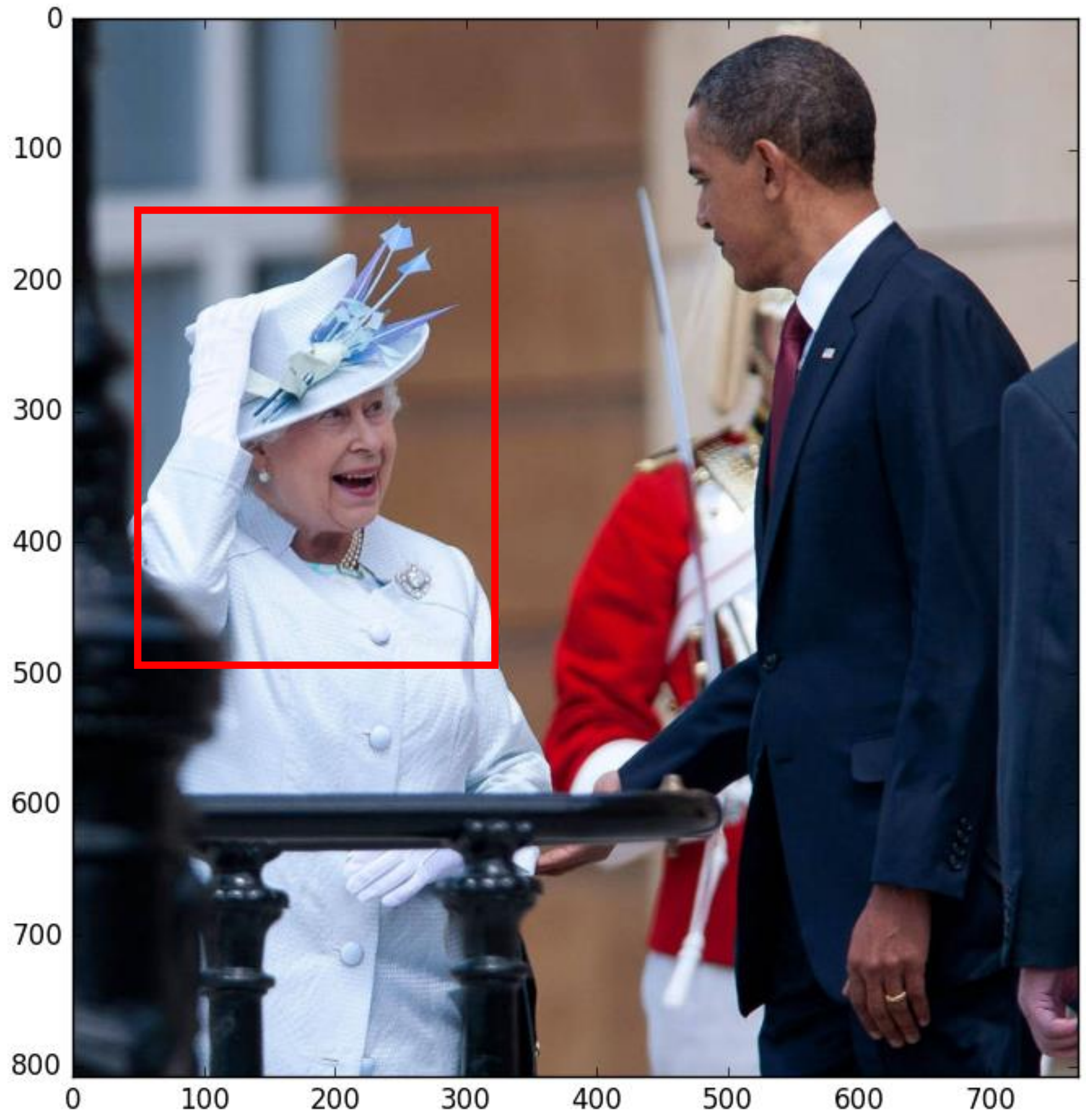
What is intuitive?
What is standard?
How would it look in black and white?
How would it be perceived by readers with colour-blindness? (4.5% of people)





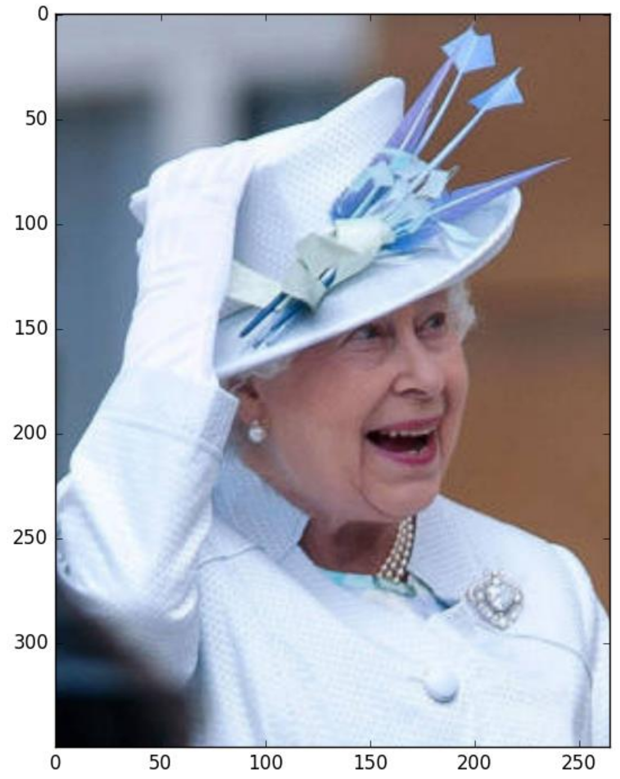
Let’s make a stamp

```
im = io.imread("thequeen.jpg")
```


0
100
200
300
400
500
600
700
800
0
100
200
300
400
500
600
700

We can slice an array just like we can slice a list, each dimension is separated by a comma

$$\texttt{cropped = im[}\overbrace{\texttt{row}_{\texttt{start}}\texttt{:row}_{\texttt{stop}}\texttt{, col}_{\texttt{start}}\texttt{:col}_{\texttt{stop}}}\texttt{]}$$



MANCHESTER 1824

The Christie NHS
NHS Foundation Trust

```
flipped = np.fliplr(cropped)
```

or

```
flipped = cropped[:,::-1]
```

A new variable

The shift along the axes as a tuple

```
shifted_image =
  interpolation.shift(im, (60, -20), mode="nearest")
```

The array we want to shift

How points outside the array should be filled. Nearest means that they take the value of the nearest point

Remember, this gave us the array dimensions as a tuple.
We are now assigning these dimensions to the variables lx and ly

```
lx, ly = shifted_image.shape
```

This gives us a grid with one horizontal array and one vertical array

```
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx * ly / 3
```

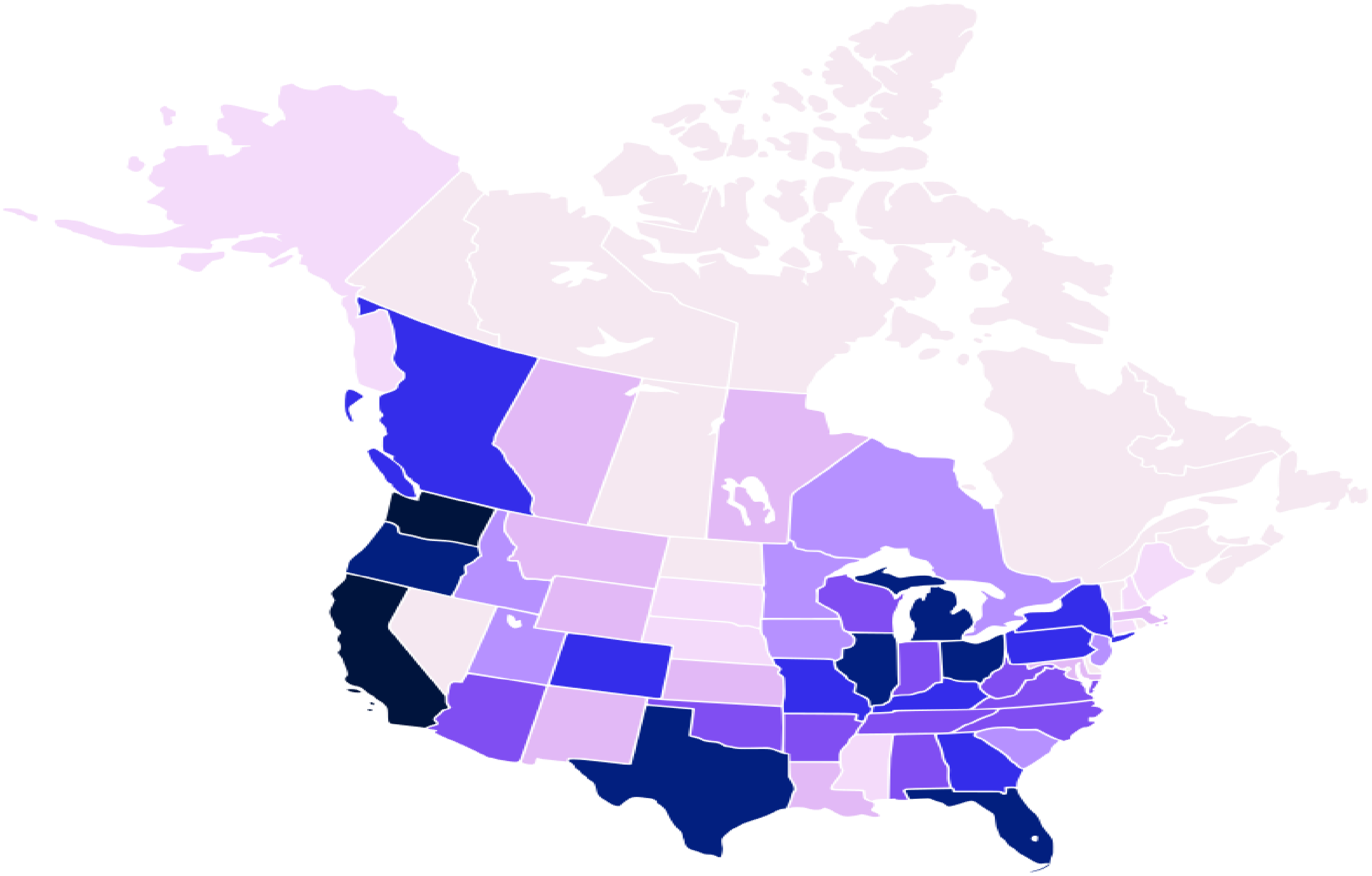
The mask is an inequality… It is also an array so it will be made out of Trues and Falses

MANCHESTER 1824

The Christie NHS
NHS Foundation Trust

With this reference, we are only picking out the array elements that are True

```
shifted_image[mask] = 0
```



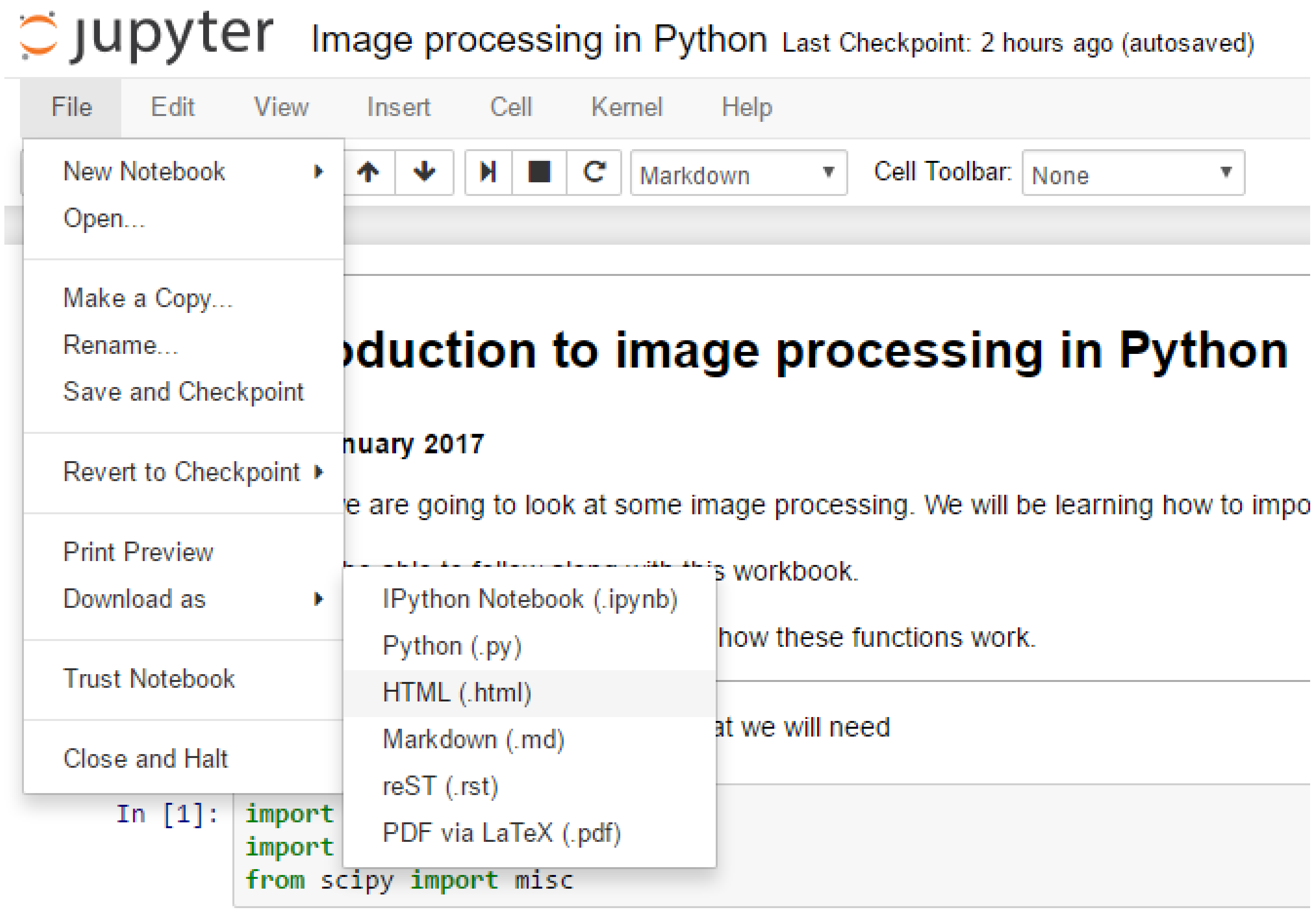
The standard deviation of the gaussian

```
blur = ndimage.gaussian_filter(im, sigma=20)
```

The image we want to blur

We could achieve something similar by convolving with a Gaussian similar to how you used convolution yesterday.

You can save your workbook to a pdf or html so that you have a copy of what you've done.



oduction to image processing in Python

nuary 2017

e are going to look at some image processing. We will be learning how to impo

s workbook.

how these functions work.

at we will need

```
In [1]: import
        import
        from scipy import misc
```

MANCHESTER 1824

The Christie NHS
NHS Foundation Trust

We wouldn’t normally write code in a notebook like we did this morning.

This afternoon, we are going to start writing code in a file which we can run.