TB_rgbNotebook

June 29, 2018

0.1 NeonScience Data Science workshop

0.2 Preinstitute Week 3

0.3 Reading RGB files in PYthon

0.3.1 Tim Bailey

```
In [1]: import sys
        sys.version
```

```
Out[1]: '3.5.5 |Anaconda custom (64-bit)| (default, May 13 2018, 21:12:35) \n[GCC 7.2.0]'
```

I had some problems with this step In the setup instructions earlier in the workshop two commands were ommitted for installing the python 3.5 version. Luckily they were there for the windows and mac versions
conda create –n p35 python=3.5 anaconda
source activate p35

```
In [2]: #import gdal
        import gdal, osr

        #not entirely sure what osr does
```

imports gdal library to open geospatial files

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import warnings
        warnings.filterwarnings('ignore')
        #unsure what the % operator does
```

Imports Numpy library for later use Numpy is a python library for doing math on arrays

```
In [4]: def RGBraster2array(RGB_geotif):
            """RGBraster2array reads in a NEON AOP geotif file and returns
            a numpy array, and header containing associated metadata with spatial information.
            --------
            Parameters
```

```
        RGB_geotif -- full or relative path and name of reflectance hdf5 file
    --------
    Returns
    --------
    array:
        numpy array of geotif values
    metadata:
        dictionary containing the following metadata (all strings):
            array_rows
            array_cols
            bands
            driver
            projection
            geotransform
            pixelWidth
            pixelHeight
            extent
            noDataValue
            scaleFactor
    --------
    Example Execution:
    --------
    RGB_geotif = '2017_SERC_2_368000_4306000_image.tif'
    RGBcam_array, RGBcam_metadata = RGBraster2array(RGB_geotif) """

    metadata = {}
    dataset = gdal.Open(RGB_geotif)
# dataset calls gdal function open and applies it the the file variable RGB_geotif
    metadata['array_rows'] = dataset.RasterYSize
    metadata['array_cols'] = dataset.RasterXSize
    metadata['bands'] = dataset.RasterCount
    metadata['driver'] = dataset.GetDriver().LongName
    metadata['projection'] = dataset.GetProjection()
    metadata['geotransform'] = dataset.GetGeoTransform()
# extracts metadata values from geotif header and assigns values to internal variables

    mapinfo = dataset.GetGeoTransform()
    metadata['pixelWidth'] = mapinfo[1]
    metadata['pixelHeight'] = mapinfo[5]

    metadata['ext_dict'] = {}
    metadata['ext_dict']['xMin'] = mapinfo[0]
    metadata['ext_dict']['xMax'] = mapinfo[0] + dataset.RasterXSize/mapinfo[1]
    metadata['ext_dict']['yMin'] = mapinfo[3] + dataset.RasterYSize/mapinfo[5]
    metadata['ext_dict']['yMax'] = mapinfo[3]
# extracts Max and Minimum coordinates from geotif metadata
    metadata['extent'] = (metadata['ext_dict']['xMin'],metadata['ext_dict']['xMax'],
                          metadata['ext_dict']['yMin'],metadata['ext_dict']['yMax'])
    raster = dataset.GetRasterBand(1)
    array_shape = raster.ReadAsArray(0,0,metadata['array_cols'],metadata['array_rows']]
    metadata['noDataValue'] = raster.GetNoDataValue()
    metadata['scaleFactor'] = raster.GetScale()

    array = np.zeros((array_shape[0],array_shape[1],dataset.RasterCount),'uint8') #pre
    for i in range(1, dataset.RasterCount+1):
        band = dataset.GetRasterBand(i).ReadAsArray(0,0,metadata['array_cols'],metadat
        band[band==metadata['noDataValue']]=np.nan
        band = band/metadata['scaleFactor']
        array[...,i-1] = band

    return array, metadata
```

I actually had problems with the previous cell I cut and paste from the workshop instruction manual in split screen I was ending up getting key errors on 'ext_dict' variable that I believe were the result of copy and paste errors related to the cut and paste function I am using Ubuntu 18.04 for this project

```
In [5]: RGB_geotif = './2017_SERC_2_368000_4306000_image.tif'
        #assigns RGB_geotif variable to a specific file
        SERC_RGBcam_array, SERC_RGBcam_metadata = RGBraster2array(RGB_geotif)
        #builds array and populates internal metadata using the preceeding function

        # it would probably be useful to change the file to a variable that could assigned
```

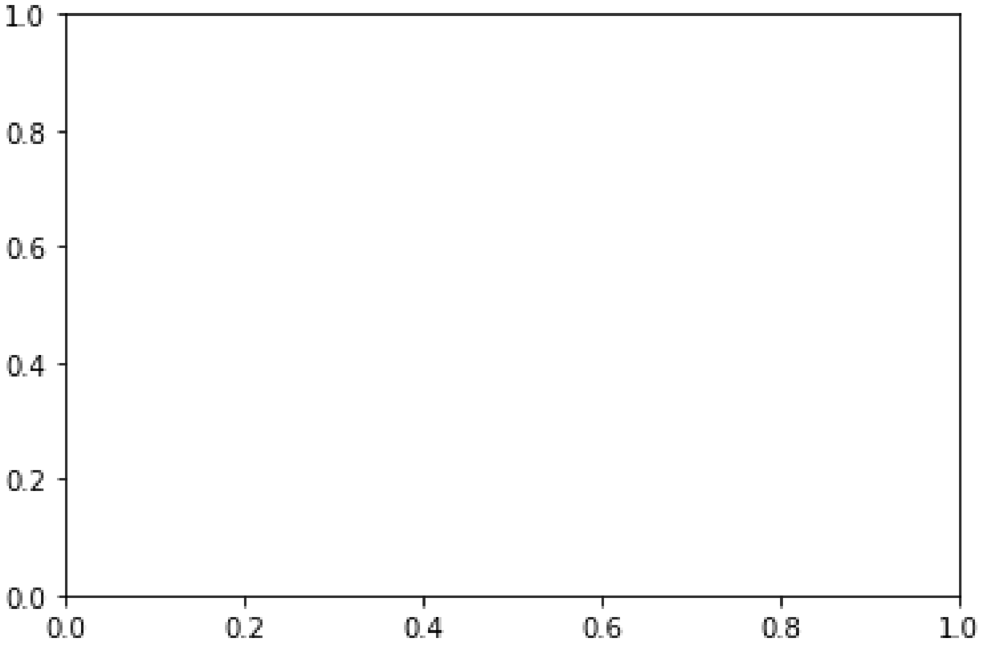
```
In [6]: SERC_RGBcam_array.shape
        # returns the dimensions of the array
```

```
Out[6]: (10000, 10000, 3)
```

```
In [7]: for key in sorted(SERC_RGBcam_metadata.keys()):
            print(key)
        # assigns the metadata
        # prints metadata variables
```

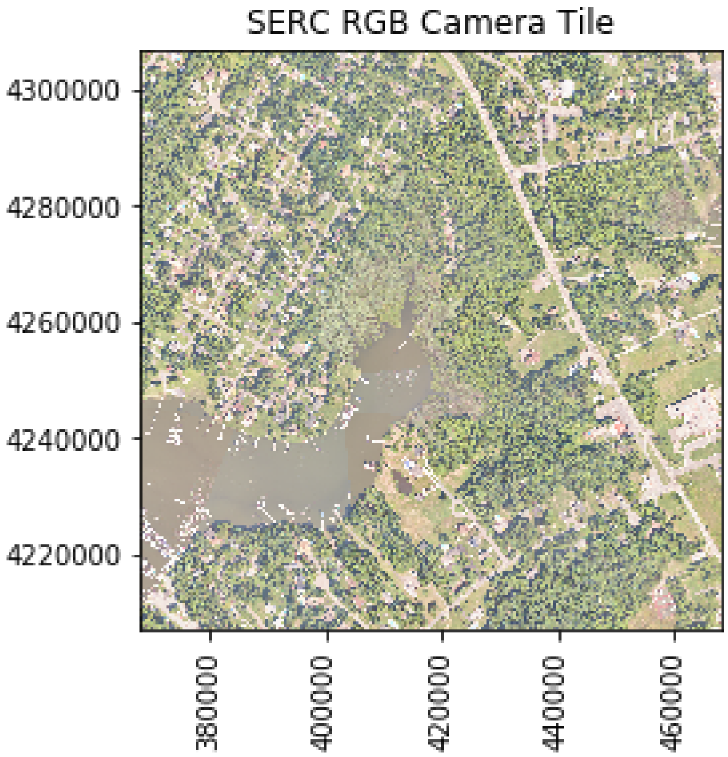
```
array_cols
array_rows
bands
driver
ext_dict
extent
geotransform
noDataValue
pixelHeight
pixelWidth
projection
scaleFactor
```

```
In [8]: def plot_band_array(band_array,
                            refl_extent,
                            colorlimit,
                            ax=plt.gca(),
                            title='',
                            cbar ='on',
                            cmap_title='',
                            colormap='spectral'):

            '''plot_band_array reads in and plots a single band or an rgb band combination of 
            --------
            Parameters
            --------
                band_array: flightline array of reflectance values, created from h5refl2array 
                refl_extent: extent of reflectance data to be plotted (xMin, xMax, yMin, yMax)
                colorlimit: range of values to plot (min,max). Best to look at the histogram o
                ax: optional, default = current axis
                title: string, optional; plot title
                cmap_title: string, optional; colorbar title
                colormap: string, optional; see https://matplotlib.org/examples/color/colormap
            --------
            Returns
                plots array of single band or RGB if given a 3-band
            --------
            Example:
            --------
            plot_band_array(SERC_RGBcam_array,
                            SERC_RGBcam_metadata['extent'],
                            (1,255),
                            title='SERC RGB Camera Tile',
                            cbar='off')'''

            plot = plt.imshow(band_array,extent=refl_extent,clim=colorlimit);
            if cbar == 'on':
                cbar = plt.colorbar(plot,aspect=40); plt.set_cmap(colormap);
                cbar.set_label(cmap_title,rotation=90,labelpad=20)
            plt.title(title); ax = plt.gca();
            ax.ticklabel_format(useOffset=False, style='plain'); #do not use scientific notati
            rotatexlabels = plt.setp(ax.get_xticklabels(),rotation=90); #rotate x tick labels 
```



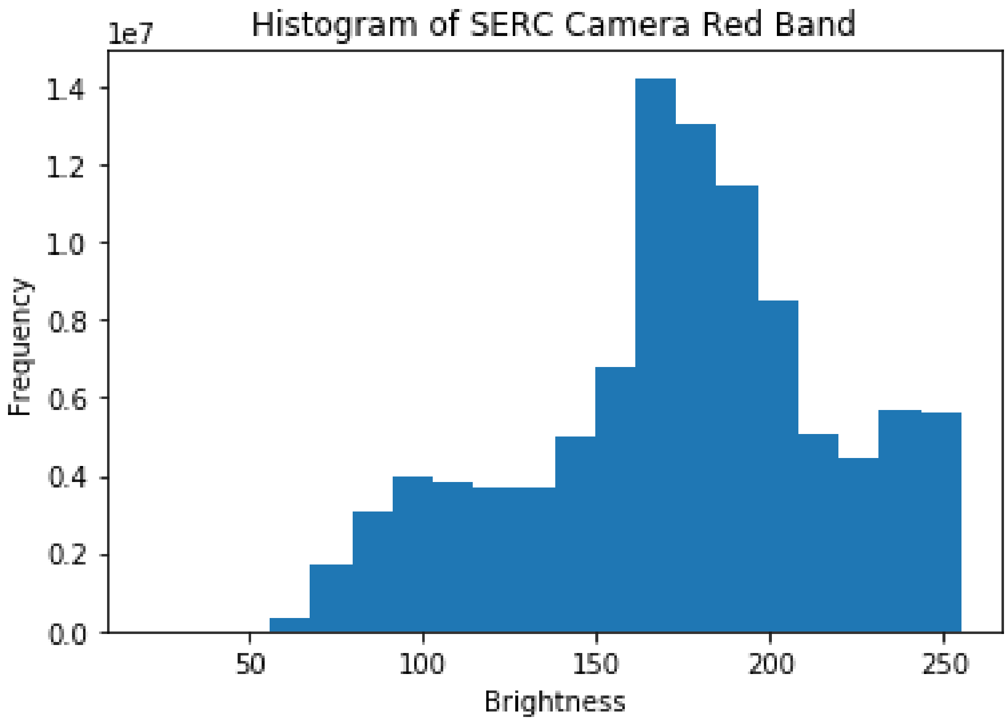
The preceeding cell defines a plotting function

```
In [9]: plot_band_array(SERC_RGBcam_array,
                        SERC_RGBcam_metadata['extent'],
                        (1,255),
                        title='SERC RGB Camera Tile',
                        cbar='off')
```

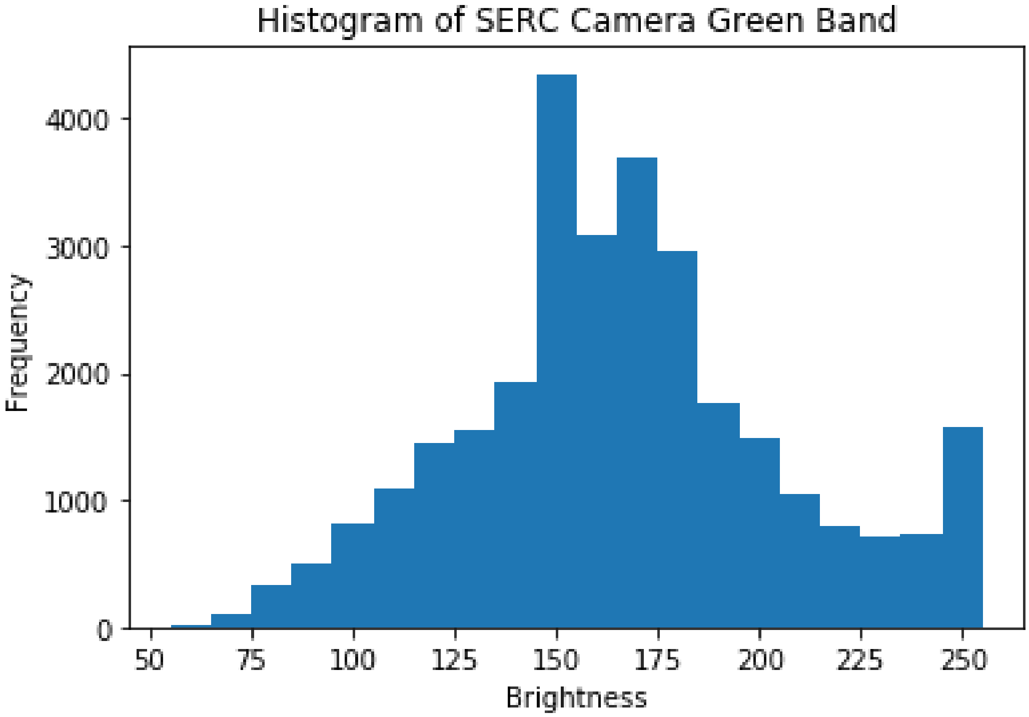


The preceeding cell plots the output of the Serc_RGBcam_array

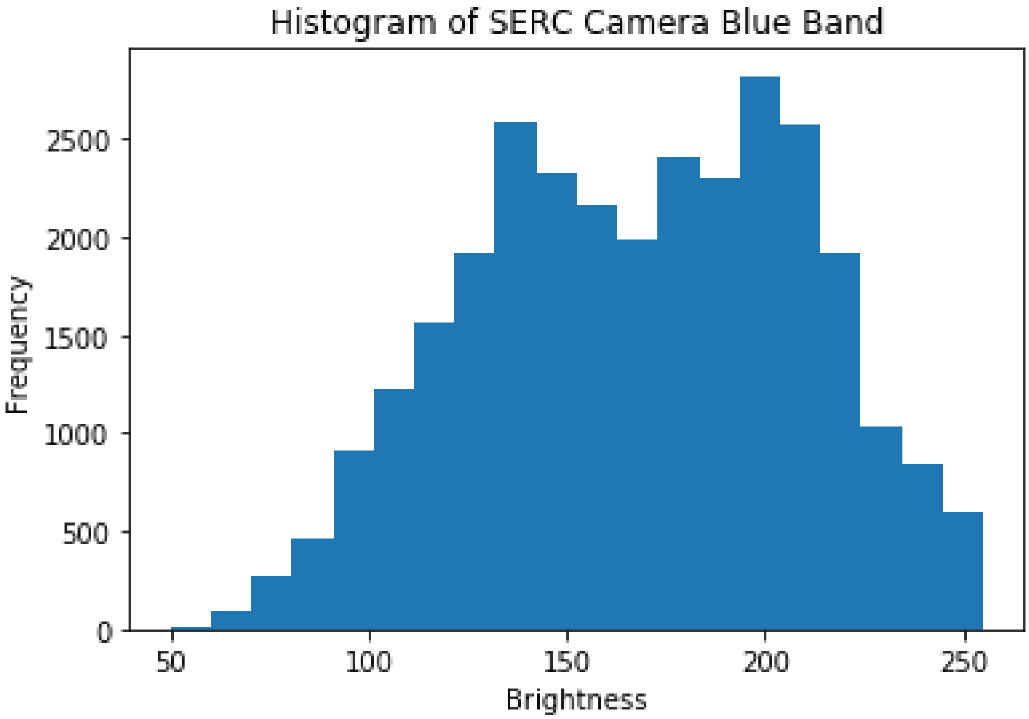
```
In [10]: plt.hist(np.ravel(SERC_RGBcam_array[:,:,0]),20);
         plt.title('Histogram of SERC Camera Red Band')
         plt.xlabel('Brightness'); plt.ylabel('Frequency')

Out[10]: Text(0,0.5,'Frequency')
```



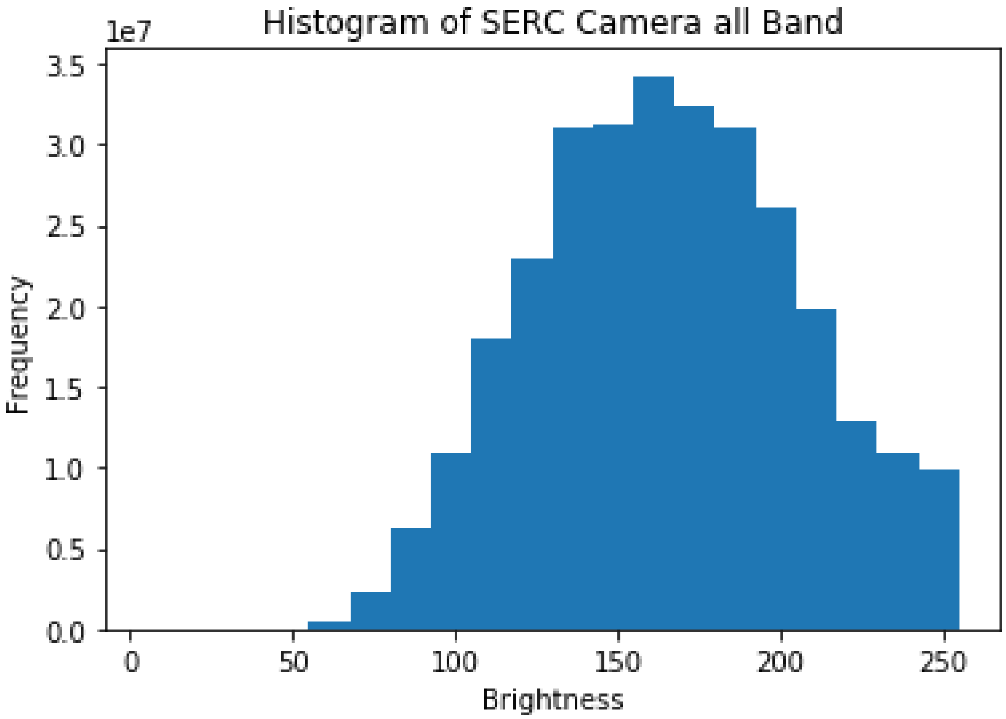
```
In [11]: # Red band above'''

In [12]: # Green band'''
         plt.hist(np.ravel(SERC_RGBcam_array[:,0,:]),20);
         plt.title('Histogram of SERC Camera Green Band')
         plt.xlabel('Brightness'); plt.ylabel('Frequency')

Out[12]: Text(0,0.5,'Frequency')
```



```
In [13]: # Blue band'''
         plt.hist(np.ravel(SERC_RGBcam_array[0,:,:]),20);
         plt.title('Histogram of SERC Camera Blue Band')
         plt.xlabel('Brightness'); plt.ylabel('Frequency')

Out[13]: Text(0,0.5,'Frequency')
```



```
In [14]: # all bands
         plt.hist(np.ravel(SERC_RGBcam_array[:,:,:]),20);
         plt.title('Histogram of SERC Camera all Band')
         plt.xlabel('Brightness'); plt.ylabel('Frequency')

Out[14]: Text(0,0.5,'Frequency')
```



```
In [15]: SERC_RGBcam_metadata['projection']

Out[15]: 'PROJCS["WGS 84 / UTM zone 18N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",63

In [16]: # Projection UTM zone 18N wgs 84

In [17]: r = np.array(SERC_RGBcam_array[:,:,0])
         np.mean(r)
         # calculate the mean of the red band using np statistical commands

Out[17]: 173.56908259

In [18]: b = np.array(SERC_RGBcam_array[0,:,:])
         # assign blue band to the array
         np.mean(b)
         #evaluate the mean of the array b

Out[18]: 168.21766666666667

In [19]: g = np.array(SERC_RGBcam_array[:,0,:])
         np.median(g)
         #   assign array
         # evaluate the median of the green band

Out[19]: 164.0
```

```
In [20]: np.var(b)
         # variance of array b
```

```
Out[20]: 1738.4926212222222
```

```
In [21]: np.std(b)
         # evaluate standard deviation of the array b
```

```
Out[21]: 41.69523499420794
```

```
In [22]: np.nanstd(b)
         #evaluate np.nanstd of array b
         # In this case np.nanstd makes no difference with np.std
```

```
Out[22]: 41.69523499420794
```

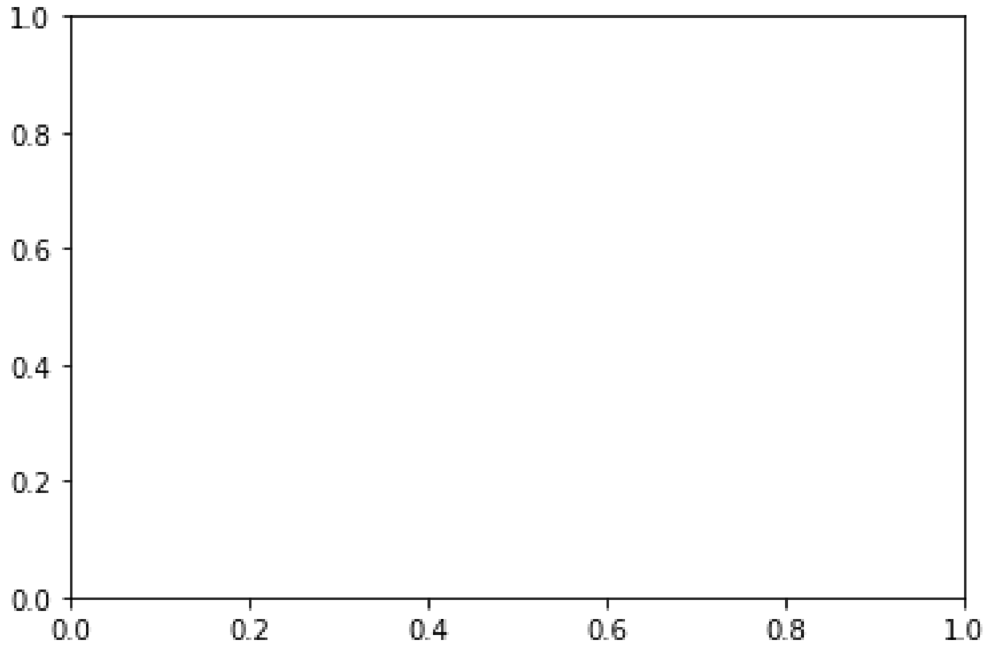
```
In [23]: SERC_RGBcam_metadata
```

```
Out[23]: {'array_cols': 10000,
          'array_rows': 10000,
          'bands': 3,
          'driver': 'GeoTIFF',
          'ext_dict': {'xMax': 468000.0,
           'xMin': 368000.0,
           'yMax': 4307000.0,
           'yMin': 4207000.0},
          'extent': (368000.0, 468000.0, 4207000.0, 4307000.0),
          'geotransform': (368000.0, 0.1, 0.0, 4307000.0, 0.0, -0.1),
          'noDataValue': None,
          'pixelHeight': -0.1,
          'pixelWidth': 0.1,
          'projection': 'PROJCS["WGS 84 / UTM zone 18N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHER
          'scaleFactor': 1.0}
```

```
In [24]: plt.hist(SERC_RGBcam_array[:,:,0]);
         plt.title('Histogram of SERC Camera Red Band')
         plt.xlabel('Brightness'); plt.ylabel('Frequency')
```

```
    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-24-18f3753da2c3> in <module>()
----> 1 plt.hist(SERC_RGBcam_array[:,:,0]);
      2 plt.title('Histogram of SERC Camera Red Band')
      3 plt.xlabel('Brightness'); plt.ylabel('Frequency')
```

```
 ~/anaconda3/envs/p35/lib/python3.5/site-packages/matplotlib/pyplot.py in hist(x, bins,
3130                      histtype=histtype, align=align, orientation=orientation,
3131                      rwidth=rwidth, log=log, color=color, label=label,
-> 3132                   stacked=stacked, normed=normed, data=data, **kwargs)
3133    finally:
3134        ax._hold = washold


 ~/anaconda3/envs/p35/lib/python3.5/site-packages/matplotlib/__init__.py in inner(ax, *a
1853                        "the Matplotlib list!)" % (label_namer, func.__name__),
1854                        RuntimeWarning, stacklevel=2)
-> 1855            return func(ax, *args, **kwargs)
1856
1857        inner.__doc__ = _add_data_doc(inner.__doc__,


 ~/anaconda3/envs/p35/lib/python3.5/site-packages/matplotlib/axes/_axes.py in hist(***fa
6528            # this will automatically overwrite bins,
6529            # so that each histogram uses the same bins
-> 6530         m, bins = np.histogram(x[i], bins, weights=w[i], **hist_kwargs)
6531            m = m.astype(float)  # causes problems later if it's an int
6532            if mlast is None:


 ~/anaconda3/envs/p35/lib/python3.5/site-packages/numpy/lib/function_base.py in histogra
 816        if weights is None:
 817            for i in arange(0, len(a), BLOCK):
--> 818             sa = sort(a[i:i+BLOCK])
 819                cum_n += np.r_[sa.searchsorted(bin_edges[:-1], 'left'),
 820                               sa.searchsorted(bin_edges[-1], 'right')]


 ~/anaconda3/envs/p35/lib/python3.5/site-packages/numpy/core/fromnumeric.py in sort(a, a
 845    else:
 846        a = asanyarray(a).copy(order="K")
--> 847 a.sort(axis=axis, kind=kind, order=order)
 848    return a
 849


 KeyboardInterrupt:
```



1 Plotted histogram above without the ravel function to bin to 20 unit

2 it definitely takes a lot longer.

```
In [25]: np.amin(b)
         #minimum blue value'''

Out[25]: 50

In [26]: np.amax(b)
         #maximum blue value'''

Out[26]: 255

In [27]: np.amin(g)
         # minimum green value'''

Out[27]: 55

In [28]: np.amax(g)
         # maximum green value'''

Out[28]: 255

In [29]: np.amin(r)
         # min red'''
```

```
Out[29]: 21
```

```
In [30]: np.amax(r)
         #max red value'''
```

```
Out[30]: 255
```

2.1 Review of histograms