Band Stacking, RGB & False Color Images, and Interactive Widgets

Using the NEON AOP Hyperspectral Python Module

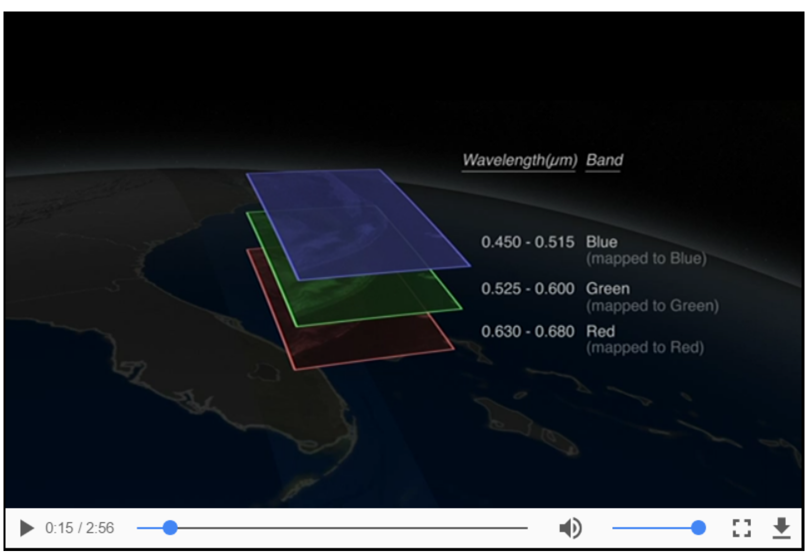
Objectives ¶

In the first lesson, we learned how to read in hdf5 data using h5py, plot a single band of a flightline of reflectance data, subset a flightline reflectance file, and plot a single band of the subsetted reflectance. We can do these tasks more efficiently using functions. In this lesson, we will load the NEON AOP Python module, which contains a series of functions to work with NEON hyperspectral data. We will also plot different combinations of bands, and learn how to create widgets to look at data more interactively.

Background

We can combine any three bands from the NEON reflectance data to make an RGB image that will depict different information about the Earth's surface. A **natural-color** image, made with bands from the red, green, and blue wavelengths looks close to what we would see with the naked eye. We can also choose band combinations from other wavelenghts, and map them to the red, blue, and green colors to highlight different features. A **false-color** image is made with one or more bands from a non-visible portion of the electromagnetic spectrum that are mapped to red, green, and blue colors. These images can display other information about the landscape that is not easily seen with a natural-color image.

The NASA Goddard Media Studio video "Peeling Back Landsat's Layers of Data" gives a good quick overview of natural and false color band combinations. Note that Landsat collects information from 11 wavelength bands, while NEON AOP hyperspectral data collects information from 426 bands!



(https://www.youtube.com/watch?v=YP0et8l_bvY)

https://www.youtube.com/watch?v=YP0et8l_bvY (https://www.youtube.com/watch?v=YP0et8l_bvY)
https://svs.gsfc.nasa.gov//vis/a010000/a011400/a011491/index.html
(https://svs.gsfc.nasa.gov//vis/a010000/a011400/a011491/index.html)

Further Reading: Check out the NASA Earth Observatory Article on "How to Interpret a False-Color Satellite Image"

https://earthobservatory.nasa.gov/Features/FalseColor/ (https://earthobservatory.nasa.gov/Features/FalseColor/)

Before we get started, let's set up our plot and warning preferences:

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

We will start by importing a module with a suite of functions that we will use in the remainder of the course. You can add to these functions or customize them to better suit your data needs. First we can see some ways to import the module into Jupyter and explore what is inside them:

- `%load module_name` - loads a module into a Jupyter Notebook cell. Allows flexibility for modifying functions.
- `%%writefile module_name.py` - writes out contents of Jupyter Notebook cell to module_name.py (overwrites if that file already exists).
- `import module_name as shortname` - imports a module, does not enable modifications.
- `help(module_name)` - lists functions in a module along associated with docstrings
- `dir(module_name)` - lists all functions and packages in a module
- `function_name?` - displays docstring associated with function_name

We will load in a module called neon_aop_refl_hdf5_functions. Download and copy this module into your current working directory, or include the path when you load it.

You can load this module into the next cell by typing:

`%load neon_aop_refl_hdf5_functions`

Alternately, if you want to import the module behind the scenes, you can type:

`import neon_aop_refl_hdf5_functions as neon`

If you choose to import the module, When you call functions from this module, you will have to first type `neon.`, eg. `neon.plot_band_array`.

In [ ]:
```
# %load neon_aop_refl_hdf5_functions
"""
Created on Mon Feb  6 16:36:10 2017

@author: bhass

neon_aop_refl_hdf5_functions contains the following functions for use in the
Remote Sensing Data Institute (June 19-25, 2017)

list_dataset (called with h5py.File.visititems):
    lists the name and location of each dataset stored in an hdf5 file

ls_dataset (called with h5py.File.visititems):
```

```
    lists name, shape, and type of each dataset stored in an hdf5 file

h5refl2array(refl_filename):
    reads in a NEON AOP reflectance hdf5 file and returns reflectance array,
    select metadata, and wavelength dataset

extract_raw_band(reflArray,reflArray_metadata,band_ind):
    extracts a single band from a reflectance array without applying the scale

    factor or data ignore value

clean_band(bandArray,reflArray_metadata):
    sets the data ignore value to NaN and applies the scale factor to a single

    reflectance band array

extract_clean_band(reflArray,reflArray_metadata,band_ind):
    extracts a single band from a reflectance array, applies the scale factor
    and sets the data ignore value to NaN

plot_band_array(band_array,refl_extent,colorlimit,ax=plt.gca(),title='', \
cmap_title='',colormap='spectral'):
    reads in and plots a single band of a reflectance array

array2raster(newRaster,reflBandArray,reflArray_metadata):
    reads in a reflectance array and associated metadata and returns a geotif
    raster named newRaster.tif

calc_clip_index(clipExtent, h5Extent, xscale=1, yscale=1):
    calculates the indices relative to a full flight line extent of a subset
    given a clip extent in UTM m (x,y)

"""

#Import Required Packages:
import numpy as np
import matplotlib.pyplot as plt
import h5py, gdal, osr, copy

def list_dataset(name,node):

    """list_dataset lists the name and location of each dataset stored in an h
df5 file.
    --------
    See Also
    --------
    ls_dataset:
        Lists name, shape, and type of each dataset stored in an hdf5 file.
    Example:
    --------
    f = h5py.File('NEON_D02_SERC_DP1_20160807_160559_reflectance.h5','r')
    f.visititems(list_dataset)"""

    if isinstance(node, h5py.Dataset):
        print(name)

def ls_dataset(name,node):
```

```
    """ls_dataset lists the name, shape, and datatype of each dataset stored i
n an hdf5 file.
    --------
    See Also
    --------
    list_dataset:
        Lists name and location of each dataset stored in an hdf5 file
    Example:
    --------
    f = h5py.File('NEON_D02_SERC_DP1_20160807_160559_reflectance.h5','r')
    f.visititems(ls_dataset)"""

    if isinstance(node, h5py.Dataset):
        print(node)

def h5refl2array(refl_filename):
    """h5refl2array reads in a NEON AOP reflectance hdf5 file and returns refl
ectance array, select metadata, and wavelength dataset.
    --------
    Parameters
        refl_filename -- full or relative path and name of reflectance hdf5 fi
le
    --------
    Returns
    --------
    reflArray:
        array of reflectance values
    metadata:
        dictionary containing the following metadata:
            EPSG: coordinate reference system code (integer)
            *bad_band_window1: [1340 1445] range of wavelengths to ignore
            *bad_band_window2: [1790 1955] range of wavelengths to ignore
            ext_dict: dictionary of spatial extent
            extent: array of spatial extent (xMin, xMax, yMin, yMax)
            mapInfo: string of map information
            *noDataVal: -9999.0
            projection: string of projection information
            *res: dictionary containing 'pixelWidth' and 'pixelHeight' values
 (floats)
            *scaleFactor: 10000.0
            shape: tuple of reflectance shape (y, x, # of bands)
        * Asterixed values are the same for all NEON AOP hyperspectral reflect
ance files processed 2016 & after.
    wavelengths:
        Wavelengths dataset. This is the same for all NEON AOP reflectance hdf
5 files.
        wavelengths.value[n-1] gives the center wavelength for band n
    --------
    This function applies to the NEON hdf5 format implemented in 2016, which a
pplies to data acquired in 2016 & 2017 as of June 2017.
    Data in earlier NEON hdf5 format is expected to be re-processed after the
 2017 flight season.
    --------
    Example
    --------
    sercRefl, sercRefl_md, wavelengths = h5refl2array('NEON_D02_SERC_DP1_20160
```

```
807_160559_reflectance.h5') """

    #Read in reflectance hdf5 file (include full or relative path if data is l
ocated in a different directory)
    hdf5_file = h5py.File(refl_filename,'r')

    #Get the site name
    file_attrs_string = str(list(hdf5_file.items()))
    file_attrs_string_split = file_attrs_string.split("'")
    sitename = file_attrs_string_split[1]

    #Extract the reflectance & wavelength datasets
    refl = hdf5_file[sitename]['Reflectance']
    reflArray = refl['Reflectance_Data']
    refl_shape = reflArray.shape
    wavelengths = refl['Metadata']['Spectral_Data']['Wavelength']

    #Create dictionary containing relevant metadata information
    metadata = {}
    metadata['shape'] = reflArray.shape
    metadata['mapInfo'] = refl['Metadata']['Coordinate_System']['Map_Info'].va
lue

    #Extract no data value & set no data value to NaN
    metadata['noDataVal'] = float(reflArray.attrs['Data_Ignore_Value'])
    metadata['scaleFactor'] = float(reflArray.attrs['Scale_Factor'])

    #Extract bad band windows
    metadata['bad_band_window1'] = (refl.attrs['Band_Window_1_Nanometers'])
    metadata['bad_band_window2'] = (refl.attrs['Band_Window_2_Nanometers'])

    #Extract projection information
    metadata['projection'] = refl['Metadata']['Coordinate_System']['Proj4'].va
lue
    metadata['epsg'] = int(refl['Metadata']['Coordinate_System']['EPSG
Code'].value)

    #Extract map information: spatial extent & resolution (pixel size)
    mapInfo = refl['Metadata']['Coordinate_System']['Map_Info'].value
    mapInfo_string = str(mapInfo);
    mapInfo_split = mapInfo_string.split(",")

    #Extract the resolution & convert to floating decimal number
    metadata['res'] = {}
    metadata['res']['pixelWidth'] = float(mapInfo_split[5])
    metadata['res']['pixelHeight'] = float(mapInfo_split[6])

    #Extract the upper left-hand corner coordinates from mapInfo
    xMin = float(mapInfo_split[3]) #convert from string to floating point numb
er
    yMax = float(mapInfo_split[4])
    #Calculate the xMax and yMin values from the dimensions
    xMax = xMin + (refl_shape[1]*metadata['res']['pixelWidth']) #xMax = left e
dge + (# of columns * resolution)",
    yMin = yMax - (refl_shape[0]*metadata['res']['pixelHeight']) #yMin = top e
dge - (# of rows * resolution)",
    metadata['extent'] = (xMin,xMax,yMin,yMax) #useful format for plotting
```

```
    metadata['ext_dict'] = {}
    metadata['ext_dict']['xMin'] = xMin
    metadata['ext_dict']['xMax'] = xMax
    metadata['ext_dict']['yMin'] = yMin
    metadata['ext_dict']['yMax'] = yMax
    hdf5_file.close

    return reflArray, metadata, wavelengths

def extract_raw_band(reflArray,reflArray_metadata,band_ind):

    """extract_raw_band extracts a single band from a reflectance array withou
t applying the scale factor or data ignore value.
     --------
    Parameters
    --------
        reflArray: array of reflectance values, created from h5refl2array func
tion
        reflArray_metadata: reflectance metadata values, created from h5refl2a
rray function
        band_ind: index of wavelength band to be extracted
    --------
    Returns
        bandArray: array of single band, without scale factor or data ignore v
alue applied.
    --------
    See Also
    --------
    clean_band:
        Applies the data ignore value and scale factor to a single band array
 of reflectance data.
    extract_clean_band:
        Extracts a single band of data from a reflectance array and applies th
e data ignore value and scale factor.
    Example:
    --------
    SERC_b56_raw = extract_raw_band(sercRefl,sercRefl_md,56) """

    bandArray = reflArray[:,:,band_ind-1].astype(np.float)
    return bandArray

def clean_band(bandArray,reflArray_metadata):

    """clean_band sets the data ignore value to NaN and applies the scale fact
or to a single reflectance band array.
     --------
    Parameters
    --------
        bandArray: array of single band of reflectance values, created from ex
tract_raw_band function
        reflArray_metadata: reflectance metadata values, created from h5refl2a
rray function
    --------
    Returns
        band_clean: array of single band, with scale factor applied and data i
gnore value set to NaN.
    --------
```

```
    See Also
    --------
    extract_raw_band:
        Extracts a single band from a reflectance array without applying the s
cale factor or data ignore value.
    extract_clean_band:
        Extracts a single band of data from a reflectance array and applies th
e data ignore value and scale factor.
    Example:
    --------
    SERC_b56_clean = clean_band(SERC_b56_raw,sercRefl_md) """

    band_clean = copy.copy(bandArray) #make a copy of the array so you don't c
hange the value of the original bandArray
    band_clean[band_clean==int(reflArray_metadata['noDataVal'])]=np.nan
    band_clean = band_clean/reflArray_metadata['scaleFactor']
    return band_clean

def extract_clean_band(reflArray,reflArray_metadata,band_ind):

    """extract_clean_band extracts a single band from a reflectance array, app
lies the scale factor and sets the data ignore value to NaN.
    --------
    Parameters
    --------
        reflArray: array of reflectance values, created from h5refl2array func
tion
        reflArray_metadata: reflectance metadata values, created from h5refl2a
rray function
        band_ind: index of wavelength band to be extracted
    --------
    Returns
        bandCleaned: array of single band, with scale factor applied and data
 ignore value set to NaN.
    --------
    See Also
    --------
    extract_raw_band:
        Extracts a single band of data from a reflectance array and applies th
e data ignore value and scale factor.
    clean_band:
        Applies the scale factor and sets the data ignore value to NaN for a s
ingle reflectance band.
    Example:
    --------
    SERC_b56_clean = extract_clean_band(sercRefl,sercRefl_md,56) """

    bandArray = reflArray[:,:,band_ind-1].astype(np.float)
    bandCleaned = copy.copy(bandArray)
    bandCleaned[bandCleaned==int(reflArray_metadata['noDataVal'])]=np.nan
    bandCleaned = bandCleaned/reflArray_metadata['scaleFactor']
    return bandCleaned

def plot_band_array(band_array,refl_extent,colorlimit,ax=plt.gca(),title='',cb
ar ='on',cmap_title='',colormap='spectral'):

    '''plot_band_array reads in and plots a single band of a reflectance array
```

```
    --------
    Parameters
    --------
        band_array: flightline array of reflectance values, created from h5ref
l2array function
        refl_extent: extent of reflectance data to be plotted (xMin, xMax, yMi
n, yMax) - use metadata['extent'] from h5refl2array function
        colorlimit: range of values to plot (min,max). Best to look at the his
togram of reflectance values before plotting to determine colorlimit.
        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/
colormaps_reference.html for list of colormaps
    --------
    Returns
        plots flightline array of single band of reflectance data
    --------
    See Also
    --------
    plot_subset_band:
        plots a subset of a full flightline reflectance band array
    Example:
    --------
    plot_band_array(SERC_b56_clean,sercRefl_md['extent'],(0,0.3),ax,title='SER
C Band 56 Reflectance',cmap_title='Reflectance',colormap='spectral') '''

    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 scientifi
c notation #
    rotatexlabels = plt.setp(ax.get_xticklabels(),rotation=90); #rotate x tick
 labels 90 degrees

def array2raster(newRaster,reflBandArray,reflArray_metadata):

    '''array2raster reads in a reflectance array and associated metadata and r
eturns a geotif raster named newRaster.tif
    --------
    Parameters
    --------
        newRaster: string, name of new geotif raster created
        reflBandArray: reflectance array to be converted to raster
        reflArray_metadata: reflectance metadata associated with reflectance a
rray (generated by h5refl2array function)
    --------
    Returns
        newRaster.tif: geotif raster created from reflectance array and associ
ated metadata
    --------
    See Also
    --------
    h5refl2array:
        reads in a NEON hdf5 reflectance file and returns the reflectance arra
```

```
y, select metadata, and the wavelength dataset
    Example:
    --------
    array2raster('SERC_b56_clean.tif',SERC_b56_clean,sercRefl_md) '''

    cols = reflBandArray.shape[1]
    rows = reflBandArray.shape[0]
    pixelWidth = float(reflArray_metadata['res']['pixelWidth'])
    pixelHeight = -float(reflArray_metadata['res']['pixelHeight'])
    originX = reflArray_metadata['ext_dict']['xMin']
    originY = reflArray_metadata['ext_dict']['yMax']

    driver = gdal.GetDriverByName('GTiff')
    outRaster = driver.Create('hopb_b56.tif', cols, rows, 1, gdal.GDT_Byte)
    outRaster.SetGeoTransform((originX, pixelWidth, 0, originY, 0,
pixelHeight))
    outband = outRaster.GetRasterBand(1)
    outband.WriteArray(reflBandArray)
    outRasterSRS = osr.SpatialReference()
    outRasterSRS.ImportFromEPSG(reflArray_metadata['epsg'])
    outRaster.SetProjection(outRasterSRS.ExportToWkt())
    outband.FlushCache()

def calc_clip_index(clipExtent, h5Extent, xscale=1, yscale=1):

    '''calc_clip_index calculates the indices relative to a full flight line e
xtent of a subset given a clip extent in UTM m (x,y)
    --------
    Parameters
    --------
        clipExtent: dictionary of extent of region
        h5Extent: dictionary of extent of h5 file (from the h5refl2array funct
ion, this corresponds to metadata['ext_dict'])
        xscale: optional, pixel size in the x-dimension, default is 1m (applic
able to NEON reflectance data)
        yscale: optional, pixel size in the y-dimension, default is 1m (applic
able to NEON reflectance data)
    --------
    Returns
        newRaster.tif: geotif raster created from reflectance array and associ
ated metadata
    --------
    Notes
    --------
    The clipExtent must lie within the extent of the h5Extent for this functio
n to work.
    If clipExtent exceets h5Extent in any direction, the function will return
 an error message.
    --------
    Example:
    --------
    clipExtent = {'xMax': 368100.0, 'xMin': 367400.0, 'yMax': 4306350.0, 'yMi
n': 4305750.0}
    calc_clip_index(clipExtent, sercRefl, xscale=1, yscale=1) '''

    h5rows = h5Extent['yMax'] - h5Extent['yMin']
    h5cols = h5Extent['xMax'] - h5Extent['xMin']
```

```
    ind_ext = {}
    ind_ext['xMin'] = round((clipExtent['xMin']-h5Extent['xMin'])/xscale)
    ind_ext['xMax'] = round((clipExtent['xMax']-h5Extent['xMin'])/xscale)
    ind_ext['yMax'] = round(h5rows - (clipExtent['yMin']-h5Extent['yMin'])/xsc
ale)
    ind_ext['yMin'] = round(h5rows - (clipExtent['yMax']-h5Extent['yMin'])/ysc
ale)

    return ind_ext

def stack_clean_bands(reflArray,reflArray_metadata,bands):

    '''stack_clean_bands generates an array of three bands, and applies the da
ta ignore value and scale factor to each band
    --------
    Parameters
    --------
        reflArray: reflectance array of dimensions (y,x,426) from which three
 bands are extracted
        reflArray_metadata: reflectance metadata associated with reflectance a
rray (generated by h5refl2array function)
        bands: indices of bands to be stacked; bands must be between 0-426 (e
g. bands=(60,30,20))
    --------
    Returns
        stackedArray: array of stacked bands
    --------
    Notes
    --------
    See Also
    --------
    h5refl2array:
        reads in a NEON hdf5 reflectance file and returns the reflectance arra
y, select metadata, and the wavelength dataset
    --------
    Example:
    --------
    sercRefl, sercRefl_md, wavelengths = h5refl2array('NEON_D02_SERC_DP1_20160
807_160559_reflectance.h5')
    RGBbands = (58,34,19)
    sercRGBarray = stack_clean_bands(sercRefl,sercRefl_md,RGBbands) '''

    band_clean_dict = {}
    band_clean_names = []

    stackedArray =
np.zeros((reflArray.shape[0],reflArray.shape[1],len(bands)),'uint8') #pre-allo
cate stackedArray matrix

    for i in range(len(bands)):
        band_clean_names.append("b"+str(bands[i])+"_refl_clean")
        band_clean_dict[band_clean_names[i]] = extract_clean_band(reflArray,re
flArray_metadata,bands[i])
        stackedArray[...,i] = band_clean_dict[band_clean_names[i]]*256

    return stackedArray
```

```
def subset_clean_band(reflArray,reflArray_metadata,clipIndex,bandIndex):

    '''subset_clean_band extracts a band from a reflectance array, subsets it
 to the specified clipIndex, and applies the no data value and scale factor
    --------
    Parameters
    --------
        reflArray: reflectance array of dimensions (y,x,426) from which multip
le bands (typically 3) are extracted
        reflArray_metadata: reflectance metadata associated with reflectance a
rray (generated by h5refl2array function)
        clipIndex: ditionary; indices relative to a full flight line extent of
 a subset given a clip extent (generated by calc_clip_index function)
        bandIndex: band number to be extracted (integer between 1-426)
    --------
    Returns
        bandCleaned: array of subsetted band with no data value set to NaN and
 scale factor applied
    --------
    See Also
    --------
    h5refl2array:
        reads in a NEON hdf5 reflectance file and returns the reflectance arra
y, select metadata, and the wavelength dataset
    calc_clip_index:
        calculates the indices relative to a full flight line extent of a subs
et given a clip extent in UTM m (x,y)
    --------
    Example:
    --------
    sercRefl, sercRefl_md, wavelengths = h5refl2array('NEON_D02_SERC_DP1_20160
807_160559_reflectance.h5')
    clipExtent = {'xMax': 368100.0, 'xMin': 367400.0, 'yMax': 4306350.0, 'yMi
n': 4305750.0}
    serc_subInd = calc_clip_index(clipExtent,sercRefl_md['ext_dict'])

    serc_b58_subset = sercRGBarray = subset_clean_band(sercRefl,sercRefl_md,se
rc_subInd,58) '''

    bandCleaned = reflArray[clipIndex['yMin']:clipIndex['yMax'],clipIndex['xMi
n']:clipIndex['xMax'],bandIndex-1].astype(np.float)
    bandCleaned[bandCleaned==int(reflArray_metadata['noDataVal'])]=np.nan
    bandCleaned = bandCleaned/reflArray_metadata['scaleFactor']

    return bandCleaned

def stack_subset_bands(reflArray,reflArray_metadata,bands,clipIndex):

    '''stack_subset_bands subsets, cleans, and stacks specified bands from a r
eflectance array
    --------
    Parameters
    --------
        reflArray: reflectance array of dimensions (y,x,426) from which multip
le bands (typically 3) are extracted
        reflArray_metadata: reflectance metadata associated with reflectance a
```

```
rray (generated by h5refl2array function)
        bands: indices of bands to be stacked; bands must be between 0-426 (e
g. bands=(60,30,20))
        clipIndex: indices relative to a full flight line extent of a subset g
iven a clip extent, (generated by calc_clip_index function)
    --------
    Returns
        stackedArray: array of subsetted, stacked bands with no data value set
 to NaN and scale factor applied
    --------
    See Also
    --------
    h5refl2array:
        reads in a NEON hdf5 reflectance file and returns the reflectance arra
y, select metadata, and the wavelength dataset
    calculcate_clip_index:
        calculates the indices relative to a full flight line extent of a subs
et given a clip extent in UTM m (x,y)
    subset_clean_band:
        extracts, subsets, and cleans a single band from a reflectance array
    --------
    Example:
    --------
    sercRefl, sercRefl_md, wavelengths = h5refl2array('NEON_D02_SERC_DP1_20160
807_160559_reflectance.h5')
    RGBbands = (58,34,19)
    clipExtent = {'xMax': 368100.0, 'xMin': 367400.0, 'yMax': 4306350.0, 'yMi
n': 4305750.0}
    serc_subInd = calc_clip_index(clipExtent,sercRefl_md['ext_dict'])

    sercRGBarray = stack_subset_bands(sercRefl,sercRefl_md,RGBbands,serc_subIn
d) '''

    subArray_rows = clipIndex['yMax'] - clipIndex['yMin']
    subArray_cols = clipIndex['xMax'] - clipIndex['xMin']

    stackedArray = np.zeros((subArray_rows,subArray_cols,len(bands)),'uint8')
#pre-allocate stackedArray matrix
    band_clean_dict = {}
    band_clean_names = []

    for i in range(len(bands)):
        band_clean_names.append("b"+str(bands[i])+"_refl_clean")
        band_clean_dict[band_clean_names[i]] = subset_clean_band(reflArray,ref
lArray_metadata,clipIndex,bands[i])
        stackedArray[...,i] = band_clean_dict[band_clean_names[i]]*256

    return stackedArray
```

We can use the h5refl2array function to read in the SERC reflectance flightline from Lesson 1. For a quick look at how to run this function, type one of the following in the next code cell:

`h5refl2array?`

`help(h5refl2array)`

In [3]:
```
sercRefl, sercRefl_md, wavelengths = h5refl2array('../data/SERC/hyperspectral/
NEON_D02_SERC_DP1_20160807_160559_reflectance.h5')
```

We can write a `for` loop to list the metadata values that this function reads in:

In [4]:
```
for item in sorted(sercRefl_md):
    print(item + ':',sercRefl_md[item])
```

```
bad_band_window1: [1340 1445]
bad_band_window2: [1790 1955]
epsg: 32618
ext_dict: {'xMin': 367167.0, 'yMin': 4300128.0, 'yMax': 4310980.0, 'xMax': 36
8273.0}
extent: (367167.0, 368273.0, 4300128.0, 4310980.0)
mapInfo: b'UTM, 1.000, 1.000, 367167.000, 4310980.000, 1.0000000000e+000, 1.0
000000000e+000, 18, North, WGS-84, units=Meters'
noDataVal: -9999.0
projection: b'+proj=UTM +zone= 18 +ellps= WGS-84 +datum= WGS-84 +units= units
=Meters +no_defs'
res: {'pixelWidth': 1.0, 'pixelHeight': 1.0}
scaleFactor: 10000.0
shape: (10852, 1106, 426)
```

Subset and Stack Bands

It is often useful to look at several bands together. We can extract and stack three bands in the red, green, and blue (RGB) spectrums to produce a color image that looks similar to what we see with our eyes. In the next part of this tutorial, we will learn to stack multiple bands and make a geotif raster from the compilation of these bands. We can see that different combinations of bands allow for different visualizations of the remotely-sensed objects and also conveys useful information about the chemical makeup of the Earth's surface.

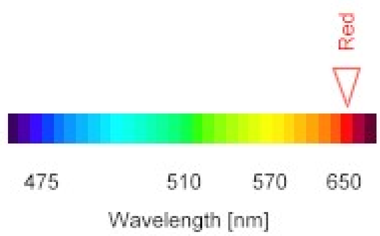
Let's use some functions from the module to start:

In [5]:
```
#Define the RGB bands (use HDFViewer for a quick look at the wavelengths corre
sponding to the bands)
RGBbands = (58,34,19) #These indexes correspond to R,G,B bands in the visible
 range of the EM spectrum

#Print the center wavelengths corresponding to these three bands:
print('Band 58 Center Wavelength = %.2f' %(wavelengths.value[57]),'nm') #Red
print('Band 34 Center Wavelength = %.2f' %(wavelengths.value[33]),'nm') #Green
print('Band 19 Center Wavelength = %.2f' %(wavelengths.value[18]),'nm') #Blue
```

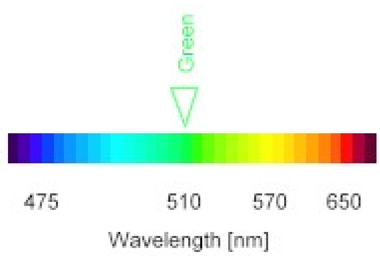
```
Band 58 Center Wavelength = 669.10 nm
Band 34 Center Wavelength = 548.91 nm
Band 19 Center Wavelength = 473.80 nm
```

We selected these bands so that they fall within the visible range of the electromagnetic spectrum (400-700 nm), where:

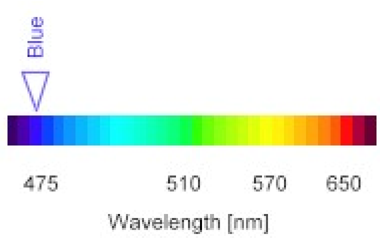
- Band 58 = 669 nm --> Red



- Band 34 = 549 nm --> Green



- Band 19 = 474 nm --> Blue



[https://science-edu.larc.nasa.gov/EDDOCS/Wavelengths_for_Colors.html (https://science-edu.larc.nasa.gov/EDDOCS/Wavelengths_for_Colors.html)](https://science-edu.larc.nasa.gov/EDDOCS/Wavelengths_for_Colors.html)

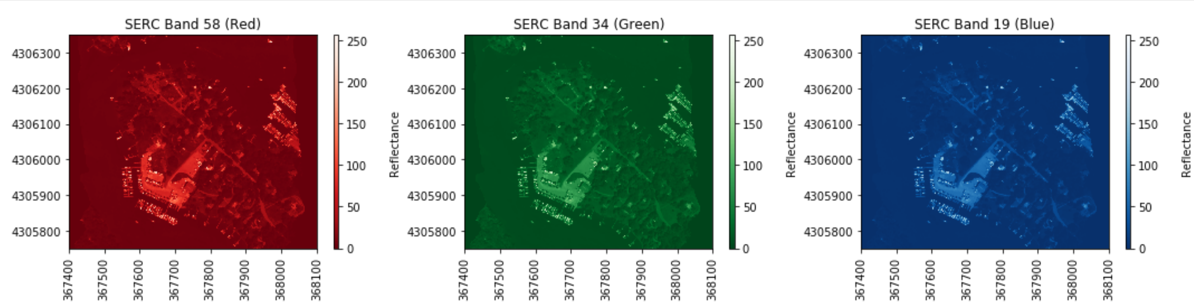
We can use the `stack_subset_bands` function to subset and stack these three bands. First we need to define the subset extent and determine the corresponding indices using the `calc_clip_index`function:

In [6]:
```
#Define the clip extent dictionary:
clipExtent = {}
clipExtent['xMin'] = 367400. #the decimal point at the end sets the data to floating point
clipExtent['xMax'] = 368100.
clipExtent['yMin'] = 4305750.
clipExtent['yMax'] = 4306350.

#Calculate the pixel indices corresponding to the extent defined above using calc_clip_index:
serc_subInd = calc_clip_index(clipExtent,sercRefl_md['ext_dict'])
print('SERC Subset Indices:',serc_subInd)

#Stack these subsetted bands using stack_subset_bands:
sercSubset_RGB = stack_subset_bands(sercRefl,sercRefl_md,RGBbands,serc_subInd)
```

```
SERC Subset Indices: {'xMin': 233, 'yMin': 4630, 'yMax': 5230, 'xMax': 933}
```

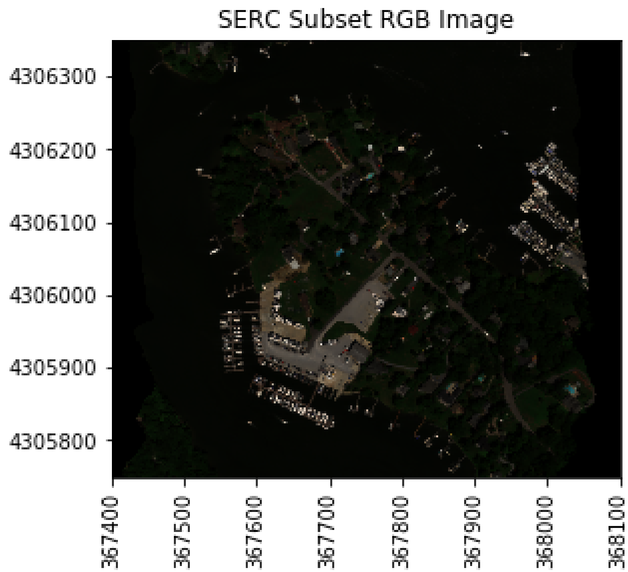
Next let's plot these three bands separately:

In [7]:
```
cmap_title='Reflectance'; colorlimit = (0,256);
clipExt = (clipExtent['xMin'],clipExtent['xMax'],clipExtent['yMin'],clipExtent['yMax'])

fig = plt.figure(figsize=(18,3.5))

ax1 = fig.add_subplot(1,3,1)
plot_band_array(sercSubset_RGB[:,:,0],clipExt,colorlimit,ax1,title='SERC Band 58 (Red)',
                     cmap_title='Reflectance',colormap='Reds_r')

ax2 = fig.add_subplot(1,3,2)
plot_band_array(sercSubset_RGB[:,:,1],clipExt,colorlimit,ax2,title='SERC Band 34 (Green)',
                    cmap_title='Reflectance',colormap='Greens_r')

ax3 = fig.add_subplot(1,3,3)
plot_band_array(sercSubset_RGB[:,:,2],clipExt,colorlimit,ax3,title='SERC Band 19 (Blue)',
                     cmap_title='Reflectance',colormap='Blues_r')
```



Finally, we can plot the three bands together:

In [8]:
```
plot_band_array(sercSubset_RGB,clipExt,(0,0.5),title='SERC Subset RGB Image',cbar='off')
```

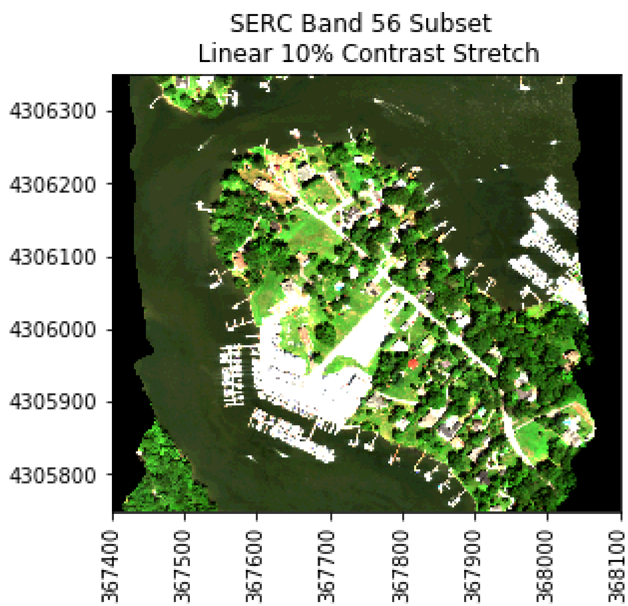


Apply Contrast Stretch & Histogram Equalization

We can also try out some basic image processing to better visualize the reflectance data using the `skimage` package, as described in Lesson 1:

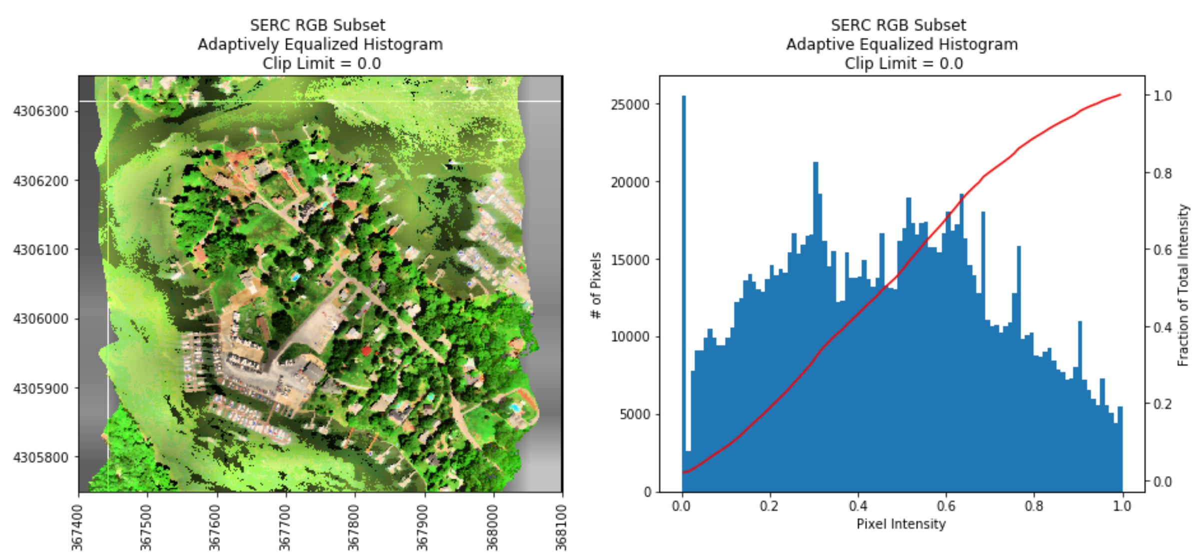
In [9]:
```
from skimage import exposure
from IPython.html.widgets import *

rgbArray = copy.copy(sercSubset_RGB)

def linearStretch(percent):
    pLow, pHigh = np.percentile(rgbArray[~np.isnan(rgbArray)], (percent,100-pe
rcent))
    img_rescale = exposure.rescale_intensity(rgbArray, in_range=(pLow,pHigh))
    plt.imshow(img_rescale,extent=clipExt)
    plt.title('SERC Band 56 Subset \n Linear ' + str(percent) + '% Contrast St
retch');
    ax = plt.gca()
    ax.ticklabel_format(useOffset=False, style='plain') #do not use scientific
 notation #
    rotatexlabels = plt.setp(ax.get_xticklabels(),rotation=90) #rotate x tick
 labels 90 degree

interact(linearStretch,percent=(0,20,1))
```

Out[9]: <function __main__.linearStretch>



In [10]:
```
#Adaptive Equalized Histogram

def adaptEqualizeHist(clip):
    img_nonan = np.ma.masked_invalid(rgbArray) #first mask the image to ignore
 the NaN values
    img_adapteq = exposure.equalize_adapthist(img_nonan, clip_limit=clip)

    fig = plt.figure(figsize=(15,6))
    ax1 = fig.add_subplot(1,2,1)

    ax1.imshow(img_adapteq,extent=clipExt,cmap='jet')
    # cbar = plt.colorbar(); cbar.set_label('Reflectance')
    plt.title('SERC RGB Subset \n Adaptively Equalized Histogram \n Clip Limit
 = ' + str(clip));
    ax1.ticklabel_format(useOffset=False, style='plain') #do not use scientifi
c notation
    rotatexlabels = plt.setp(ax1.get_xticklabels(),rotation=90) #rotate x tick
 labels 90 degree

    # Display histogram (100 bins)
    bins = 100
    ax_hist = fig.add_subplot(1,2,2)
    ax_hist.hist(img_adapteq.ravel(),bins); #np.ravel flattens an array into o
ne dimension
    plt.title('SERC RGB Subset \n Adaptive Equalized Histogram \n Clip Limit =
 ' + str(clip));
    ax_hist.set_xlabel('Pixel Intensity'); ax_hist.set_ylabel('# of Pixels')

    # Display cumulative distribution
    ax_cdf = ax_hist.twinx()
    img_cdf, bins = exposure.cumulative_distribution(img_adapteq,bins)
    ax_cdf.plot(bins, img_cdf, 'r')
    ax_cdf.set_ylabel('Fraction of Total Intensity')

interact(adaptEqualizeHist,clip=(0,1,.05))
```

Out[10]: <function __main__.adaptEqualizeHist>

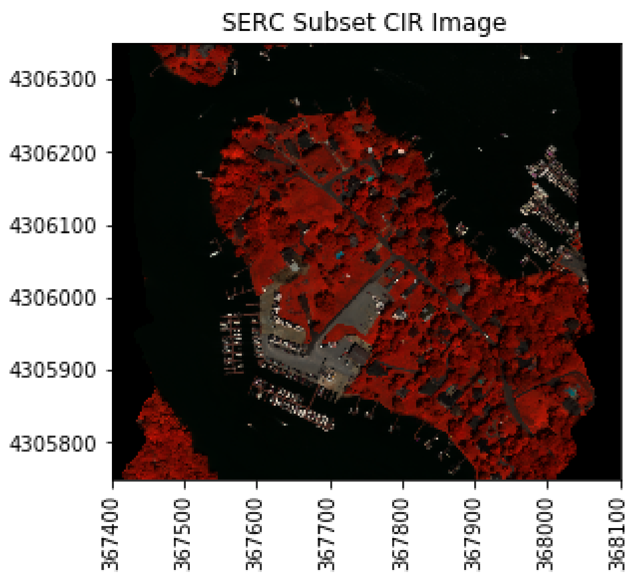


Color Infrared (CIR) Image

Finally, we'll make a color-infrared (CIR) image, where we will use the same Green and Blue bands as in the RGB array, but we'll replace the Red band with one in the Infrared range of the electromagnetic spectrum ():

In [11]:
```
CIRbands = (90,34,19)
print('Band 90 Center Wavelength = %.2f' %(wavelengths.value[89]),'nm')
print('Band 34 Center Wavelength = %.2f' %(wavelengths.value[33]),'nm')
print('Band 19 Center Wavelength = %.2f' %(wavelengths.value[18]),'nm')

sercSubset_CIR = stack_subset_bands(sercRefl,sercRefl_md,CIRbands,serc_subInd)
plot_band_array(sercSubset_CIR,clipExt,(0,0.5),title='SERC Subset CIR Image',c
bar='off')
```

```
Band 90 Center Wavelength = 829.34 nm
Band 34 Center Wavelength = 548.91 nm
Band 19 Center Wavelength = 473.80 nm
```



On Your Own: False Color Image

We can also create an image from bands outside of the visible spectrum. An image containing one or more bands outside of the visible range is called a **false-color image**. Here we'll use bands with wavelengths in two Short Wave Infrared (SWIR) bands (1100-3000 nm) and one red band (669 nm).

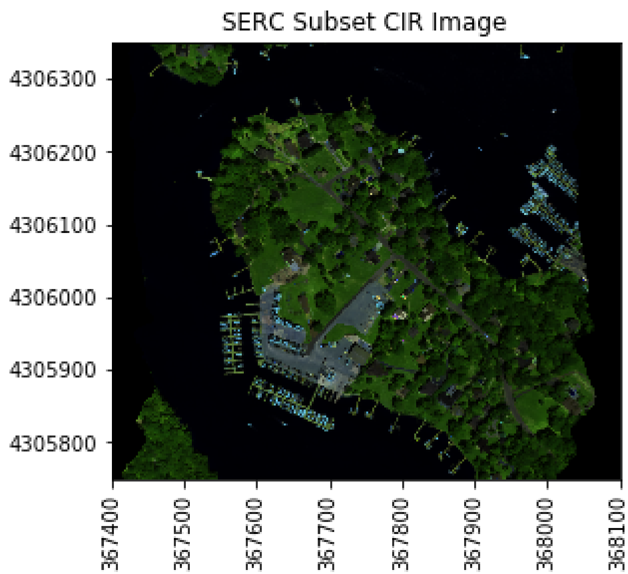
For more information about non-visible wavelengths, false color images, and some frequently used false-color band combinations, refer to NASA's Earth Observatory page:

https://earthobservatory.nasa.gov/Features/FalseColor/
(https://earthobservatory.nasa.gov/Features/FalseColor/)

Example solution below:

In [12]:
```
FCIbands = (363,246,58)
print('Band 90 Center Wavelength = %.2f' %(wavelengths.value[362]),'nm')
print('Band 34 Center Wavelength = %.2f' %(wavelengths.value[246]),'nm')
print('Band 19 Center Wavelength = %.2f' %(wavelengths.value[58]),'nm')

sercSubset_FCI = stack_subset_bands(sercRefl,sercRefl_md,FCIbands,serc_subInd)
plot_band_array(sercSubset_FCI,clipExt,(0,0.5),title='SERC Subset CIR Image',c
bar='off')
```

```
Band 90 Center Wavelength = 2196.45 nm
Band 34 Center Wavelength = 1615.56 nm
Band 19 Center Wavelength = 674.11 nm
```



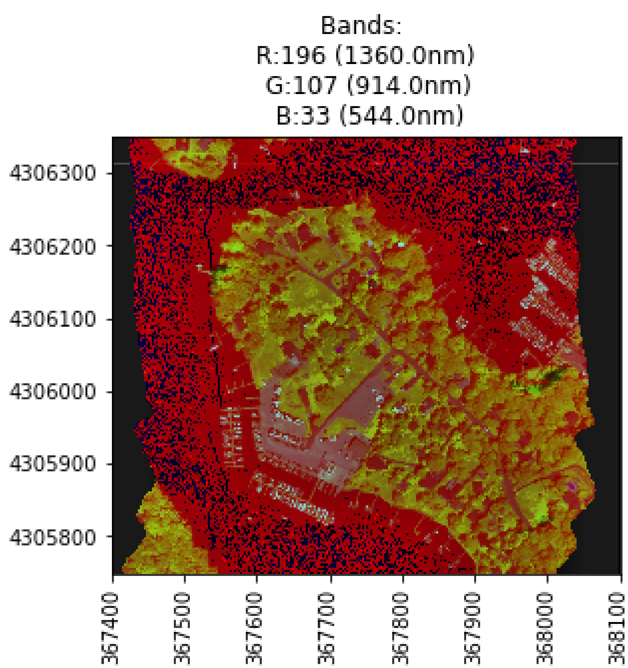
Try out Different RGB Band Combinations Interactively

Now that we have made a couple different band combinations, we can create a Python widget to explore different combinations of bands in the visible and non-visible portions of the spectrum.

In [13]:
```
from IPython.html.widgets import *

#Subset the SERC reflectance array using the indices determined from calc_clip
_index
serc_subArray =
sercRefl[serc_subInd['yMin']:serc_subInd['yMax'],serc_subInd['xMin']:serc_subI
nd['xMax'],:]
```

In [14]:
```
array = copy.copy(serc_subArray)
Refl_md = copy.copy(sercRefl_md)
```

In [15]:
```
def RGBplot_widget(R,G,B):

    #Pre-allocate array  size
    rgbArray = np.zeros((array.shape[0],array.shape[1],3), 'uint8')

    Rband = array[:,:,R-1].astype(np.float)
    Rband_clean = clean_band(Rband,Refl_md)

    Gband = array[:,:,G-1].astype(np.float)
    Gband_clean = clean_band(Gband,Refl_md)

    Bband = array[:,:,B-1].astype(np.float)
    Bband_clean = clean_band(Bband,Refl_md)

    rgbArray[..., 0] = Rband_clean*256
    rgbArray[..., 1] = Gband_clean*256
    rgbArray[..., 2] = Bband_clean*256

    # Apply Adaptive Histogram Equalization to Improve Contrast:

    img_nonan = np.ma.masked_invalid(rgbArray) #first mask the image
    img_adapteq = exposure.equalize_adapthist(img_nonan, clip_limit=0.10)

    plot = plt.imshow(img_adapteq,extent=clipExt);
    plt.title('Bands: \nR:' + str(R) + ' (' + str(round(wavelengths.value[R-
1])) +'nm)'
              + '\n G:' + str(G) + ' (' + str(round(wavelengths.value[G-1])) +
 'nm)'
              + '\n B:' + str(B) + ' (' + str(round(wavelengths.value[B-1])) +
 'nm)');
    ax = plt.gca(); ax.ticklabel_format(useOffset=False, style='plain')
    rotatexlabels = plt.setp(ax.get_xticklabels(),rotation=90)

interact(RGBplot_widget, R=(1,426,1), G=(1,426,1), B=(1,426,1))
```



References

Kekesi, Alex et al. "NASA | Peeling Back Landsat's Layers of Data". https://svs.gsfc.nasa.gov/vis/a010000/a011400/a011491/ (https://svs.gsfc.nasa.gov/vis/a010000/a011400/a011491/). Published on Feb 24, 2014.

Riebeek, Holli. "Why is that Forest Red and that Cloud Blue? How to Interpret a False-Color Satellite Image" https://earthobservatory.nasa.gov/Features/FalseColor/ (https://earthobservatory.nasa.gov/Features/FalseColor/)